**US Army Corps
of Engineers**
Waterways Experiment
Station

Microbial Degradation of Volatile Anthropogenic Organic Chemicals

by Martin Alexander, Francois Roch, Cornell University

DTIC
SELECTED
APR 1 7 1995
G
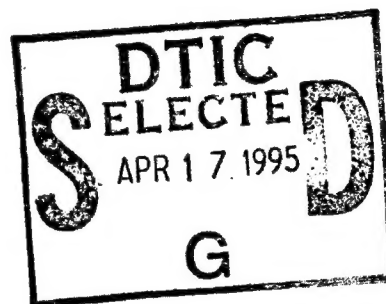
19950414 082

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.

Microbial Degradation of Volatile Anthropogenic Organic Chemicals

by Martin Alexander, Francois Roch

Department of Soil, Crop, and Atmospheric Sciences
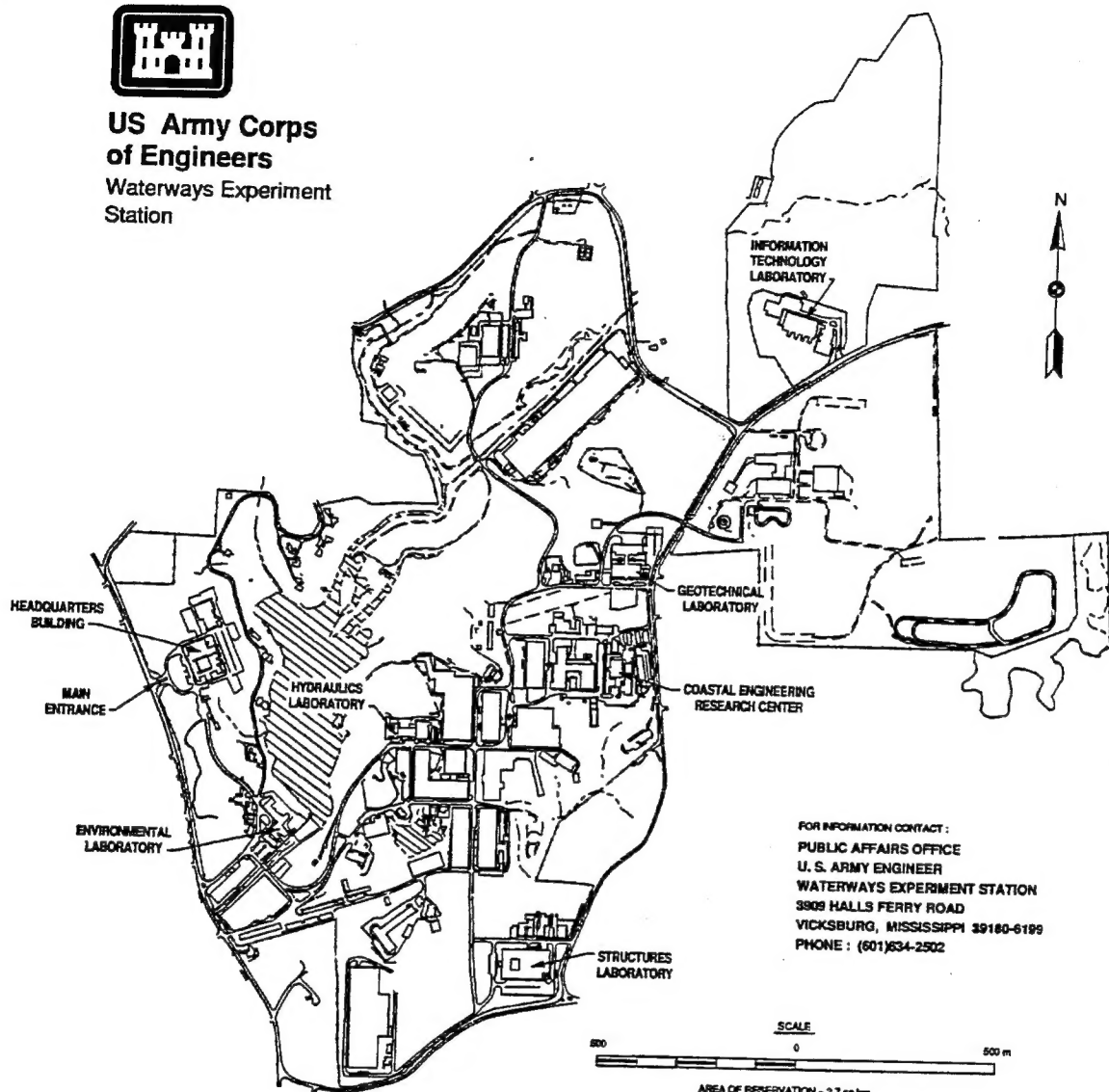Cornell University
Ithaca, NY 14853

Accesion For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification _____	
By _____	
Distribution /	
Availability Codes	

Dist	Avail and / or Special
A-1	

Final report

Approved for public release; distribution is unlimited

**US Army Corps
of Engineers**
Waterways Experiment
Station

Contents

List of Figures

Preface

This report should be cited as follows:

Alexander, M., and Roch, F. (1995). "Microbial degradation of volatile anthropogenic organic chemicals," Miscellaneous Paper EL-95-2, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

1 Introduction

Background

Many volatile compounds are released as a consequence of Army and civilian activities. These compounds may originate from air stripping of contaminated aquifers, manufacturing or processing procedures, or the use of products containing volatile constituents. Some of these compounds are innocuous, but others are toxic or objectionable because of their foul odors. Many, and possibly nearly all, of the organic compounds emitted into the air are biodegradable, and thus it is likely that practical technologies can be developed to bring about their biodegradation. Indeed, a variety of processes have been developed to destroy organic compounds in waste gases when those compounds support microbial growth. However, many of the organic pollutants of interest to the Army do not support growth but are rather cometabolized.

Trichloroethylene (TCE)[1] is an example of such a compound. TCE is widely used in the dry-cleaning industry and in the industrial industry as a degreasing solvent. It is a suspected human carcinogen and a widespread contaminant in soil and groundwater.

TCE is quite resistant to microbial degradation, and no known microorganism is able to use TCE as a carbon and energy source. However, TCE was found to be a substrate for several oxygenases of low substrate specificity, including ammonia mono-oxygenase, toluene dioxygenase, and soluble methane mono-oxygenase. The last enzyme is responsible for the oxidation of methane to methanol in bacteria known as methanotrophs.

Methanotrophs are gram negative, strict aerobes able to grow on methane as a sole source of carbon and energy. The degradation pathway of methane involves its oxidation to methanol, formaldehyde, formate, and carbon dioxide. Carbon necessary for cell metabolism is assimilated at the level of formaldehyde by one of two possible pathways, the ribulose monophosphate (RuMP) pathway and the serine pathway (Figure 1). All

---

[1] For convenience, symbols and abbreviations are listed in the Notation (Appendix B).
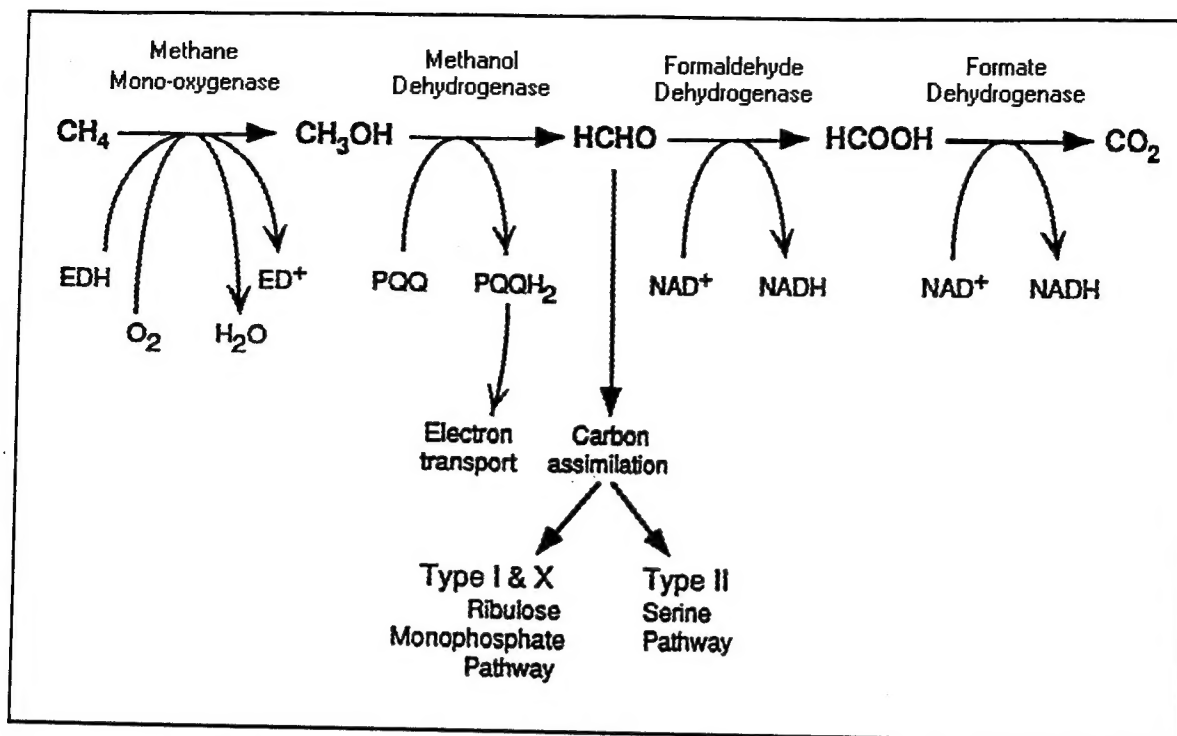
Figure 1. Pathways of methane metabolism by methanotrophic bacteria. EDH is the electron donor of methane mono-oxygenase. It is NADH if the enzyme is soluble methane mono-oxygenase, but its identity is unknown with particulate methane mono-oxygenase. PQQ is the cofactor pyrrolo-quinoline quinone

methanotrophs are able to form resting structures called cysts or exospores, and the bacterial cells have complex systems of internal membranes when grown with methane (Stanier et al. 1987).

Methanotrophs can be divided into three groups, type I, type II, and type X, depending on several physiological and biochemical traits, including the structure of their internal membrane and the pathway of formaldehyde assimilation. Type X was recently added to accommodate strains of *Methylococcus capsulatus*, which are the only group of methanotrophs capable of autotrophic CO_2 fixation (Green 1992). These strains were previously categorized as type I methanotrophs (Whittenbury and Krieg 1984).

The enzyme methane mono-oxygenase (MMO) catalyzes the oxidation of methane to methanol by the addition of one atom of oxygen (from molecular oxygen) to methane. The second atom of molecular oxygen is reduced to H_2O at the expense of reducing power. There are two types of MMO. One is membrane bound and is called particulate MMO (pMMO). The other is soluble in the cytoplasm and is called soluble MMO (sMMO). Particulate MMO may contain a copper-containing cofactor, and it is synthesized preferentially to sMMO when copper is present in sufficient concentration. In the absence of copper, some methanotrophs are able to synthesize sMMO, an enzyme which contains an iron cofactor (Fox et al. 1989).

2

Soluble MMO has a low substrate specificity and is able to oxidize (i.e., cometabolize) a variety of other compounds including TCE. Particulate MMO is also able to oxidize TCE but at a much slower rate (DiSpirito et al. 1992). Not all methanotrophs are able to synthesize sMMO when copper is absent. Type II and type X methanotrophs are able to synthesize sMMO, but type I methanotrophs are not. The only known exception is the type I methanotroph *Methylomonas methanica* 68-1 (Koh, Bowman, and Sayler 1993). The studies discussed herein deal with conditions under which TCE is rapidly degraded. Therefore, the methanotrophs are grown under copper-limited conditions to favor the synthesis of sMMO.

Several products are obtained from the oxidation of TCE by sMMO. The primary oxidation products of TCE are assumed to be TCE epoxide and chloral (2,2,2 - trichloroacetaldehyde). TCE epoxide is then spontaneously and rapidly hydrolyzed to carbon monoxide, formate, glyoxylate, and dichloroacetate (Fox et al. 1990). The proportion of the different products varies depending on the species of methanotroph, and the fate of these products is discussed in the following paragraphs.

Chloral was found to be biologically transformed to trichloroethanol and trichloroacetic acid by *Methylosinus trichosporium* OB3b, a type II methanotroph, presumably by the action of the enzyme methanol dehydrogenase (Newman and Wackett 1991). Carbon monoxide is a substrate of sMMO, and its oxidation to carbon dioxide was found to inhibit TCE oxidation both by exerting a demand for reducing factor (needed as cosubstrate for sMMO catalysis) and through competitive inhibition (Henry and Grbic-Galic 1991a). Formate is the usual substrate of the enzyme formate dehydrogenase in methanotrophs, and it may be the only degradation product of TCE that provides the cell with any benefit.

An experiment on the biological oxidation of radiolabeled [1,2-^{14}C]TCE by *Methylocystis* sp. strain M (a type II methanotroph) indicated that dichloroacetate and trichloroacetate accumulated in the medium. Glyoxylate accumulated during the mid-log phase and then was oxidized; carbon monoxide did not accumulate, but ^{14}C-carbon dioxide did (Uchiyama et al. 1992). Nakajima et al. (1992) found that strain M was not able to utilize glyoxylate as the sole carbon source, and they suggest that glyoxylate was assimilated by cooxidation in strain M. When ^{14}C-TCE was degraded by a mixed culture from which strain M was isolated, dichloroacetate and glyoxylate were completely converted to CO_2, but trichloroacetate was somewhat more resistant to further degradation (Uchiyama et al. 1992).

Like many mono-oxygenases (Walsh 1979), sMMO requires a reducing cofactor (NADH) to catalyze the oxidation of a substrate by molecular oxygen (Fox et al. 1989). As a consequence, the oxidation of TCE or carbon monoxide by sMMO depletes the energy reserves of the cells.

In methanotrophs, the oxidation of formate to carbon dioxide is always coupled to the reduction of NAD^+ to NADH (Anthony 1982), but methanotrophs do not seem to grow on formate (Whittenbury and Krieg 1984). Therefore, formate can be considered to be a source of reducing factor but not a growth substrate. Alvarez-Cohen and McCarty (1991b) reported that the addition of formate resulted in increased initial rates of TCE transformation and an elevated total transformation capacity of TCE by a mixed methanotrophic culture in the absence of methane (i.e., by resting cells). These effects presumably occurred because the addition of formate slowed the depletion of endogenous energy reserves of the cells, a depletion partially resulting from the use of reducing power during TCE oxidation.

Similarly, Henry and Grbic-Galic (1991b) reported that during methane starvation, the addition of formate increased the rates of TCE transformation by *Methylomonas* species, but this was not observed in experiments with a mixed culture of methanotrophs. These authors observed lipid storage granules in several of the cells and suggested that they served as an endogenous source of electrons for TCE oxidation during methane starvation. Earlier studies by Whittenbury, Phillips, and Wilkinson (1970) showed that such storage granules in methanotrophs contained polyhydroxybutyrate (PHB). In addition, Henrysson and McCarty (1993) found a positive correlation between the PHB content of resting mixed cultures of methanotrophs and the transformation rate of TCE as well as between the amount of PHB and the activity of sMMO (assayed by measuring the oxidation rate of naphthalene by sMMO).

In addition to the consumption of reducing power, the oxidation of TCE is detrimental to the bacterial cell. Studies on the purified sMMO of *Methylosinus trichosporium* OB3b by Fox et al. (1990) demonstrated a decrease in sMMO activity following TCE oxidation. An oxidation product of TCE and not TCE itself was shown to be responsible for the inactivation. Each component of the enzyme became radiolabeled following the oxidation of 14C-TCE, and it was suggested that a diffusible hydrolysis product of TCE epoxide formed covalent bonds with the components of sMMO. Similar results were obtained by Oldenhuis et al. (1991), who showed that cells of *Methylosinus trichosporium* OB3b became inactivated following the oxidation of ^{14}C-TCE. Various cell proteins became covalently radiolabeled, including the hydroxylase component of sMMO.

Objectives

The objectives of the research were as follows:

a. Devise a cometabolic system to bring about the degradation of volatlile organic compounds not supporting microbial growth.

4

b. Develop a means for maintaining the microbial population on a solid support or in a liquid system.

c. Determine the organic products that are generated during the cometabolic biodegradation and establish a means to destroy those compounds.

d. Establish whether there is a need for specialized microorganisms to degrade compounds sorbed to solid supports and establish a means for promoting the development of those microorganisms on the solid phase.

e. Determine if there is a threshold for the biodegradation of volatile organic compounds that are cometabolized and that are degraded as a consequence of microbial growth.

The following paragraphs briefly summarize how the above objectives were approached.

Consistent with the first objective, the test compound was TCE, and its cometabolic transformation was performed by methanotrophs (and, at an early phase of the study, by propane oxidizers).

There were only a few problems associated with maintaining the microbial population in a liquid system, and the biodegradation of TCE in the presence of several solid supports was investigated.

Some analyses were performed to detect potential volatile products generated during the cometabolic biodegradation of TCE.

Attempts were made to study the biodegradation of TCE sorbed on granular activated carbon (GAC), but the analytical method was not reliable. Additional studies were conducted on a procedure that consisted of extracting TCE from GAC using methanol and of the biodegradation by methanotrophs of TCE dissolved in methanol.

A study was conducted to determine if there was a threshold concentration below which TCE will not be cometabolized by different inocula of methanotrophs.

In addition to the objectives, to facilitate the upscaling of a bioreactor, an intensive study was conducted on the kinetics of TCE degradation by methanotrophs, including the development of a computer program. However, because of time constraints, only preliminary designs were made for a bioreactor.

2 Materials and Methods

Chemicals

Trichloroethylene (TCE) (>99 percent pure, spectrophotometric grade), 1,1,1-trichloroethane (TCA) (99 percent), dichlorodimethylsilane (99 percent), and pentane (>99 percent) were from Aldrich Chemical Co., Milwaukee, WS. Methane (grade 4.0), propane (grade 3.7), nitrogen (grade 5.0), air (grade 0.1), and hydrogen (grade 5.0) were from Airco, The BOC Group, Inc., Murray Hill, NJ. Hexane (grade "optima") was from Fisher Scientific Co, Rochester, NY.

Mineral Salts

Following is the aqueous solution of mineral salts contained in grams per liter of distilled water: $MgSO_4 \cdot 7H_2O$, 0.5; $Ca(NO_3)_2 \cdot 4H_2O$, 0.3; NH_4NO_3, 0.2; K_2HPO_4, 0.7; NaH_2PO_4, 0.2; $Fe(NO_3)_3 \cdot 5H_2O$, 0.007. The concentration of trace nutrients in the solution was in milligrams per liter: $ZnSO_4 \cdot 7H_2O$, 0.64; $Na_2MoO_4 \cdot H_2O$, 0.16; $MnSO_4 \cdot H_2O$, 0.48; H_3BO_3, 0.2; $Co(NO_3)_2 \cdot 6H_2O$, 0.8; $Ni(NO_3)_2 \cdot 6H_2O$, 0.08; KI, 0.2. In addition, 0.024 ml of HNO_3 (70 percent) was added per liter of solution.

The procedure used to prepare the aqueous solution of mineral salts was as follows. The distilled water and $MgSO_4 \cdot 7H_2O$, $Ca(NO_3)_2 \cdot 4H_2O$, and NH_4NO_3 were sterilized together by autoclaving at 121 °C for 25 min. Then phosphate salts, which were sterilized separately by autoclaving, were added in the solution. Iron was then added to the salts solution from a stock that had been sterilized by filtration through a 0.2-μm pore-size filter contained in a Nalgene disposable filter unit (Nalge, Inc., Rochester NY). The stock iron solution contained HNO_3 at a concentration of 0.066 M. The stock solution of trace nutrients was prepared by adding the different salts to sterilized distilled water containing 0.066 M of HNO_3. It was expected that the low pH of the solution killed most of the microorganisms that may be present; nevertheless, this solution may not have been sterile. Autoclaving of the solution containing trace nutrients was avoided to prevent the possible precipitation of some of the salts. In addition, this

6

stock was not filter-sterilized because some of the salts might have been retained by the filter (which was observed with iron and some filters). The stock of trace nutrients was clear following its preparation, but it turned yellow after 1 week. Its chemical composition thus may have changed with time.

Microorganisms

Enrichments of microorganisms able to degrade propane, pentane, and hexane were obtained from soil. The methane-degrading enrichment designated Ma was also obtained from soil. Enrichments Mb and Mc, which are able to grow on methane, were obtained from an expanded bed bioreactor fed with methane and were provided by W. J. Jewell of Cornell University. From these two enrichments, five strains able to grow on methane were isolated after purification on solid medium (strains M1, M2, M3, M4, and M5). The methanotroph *Methylococcus capsulatus* (Bath) (MCB) and an unidentified but pure methanotrophic culture (Mx) were obtained from Cornell University.

The pure bacterial cultures and enrichments able to grow on methane and propane were grown in test tubes loosely closed with screw caps and placed inside hermetically closed jars (BBL Gas Pack System from VWR Scientific Rochester, NY) containing a mixture of about 30-percent methane and 70-percent air. These jars were incubated at 30 °C on a rotary shaker. The test tubes usually became turbid after 2 days of incubation when the carbon source was methane, but they only became turbid after about 6 days when the carbon source was propane.

The purity of the cultures was checked periodically by streaking them on agar plates containing mineral salts and 1.6 percent of Bacto-agar (Difco Laboratories, Detroit, MI). These plates were placed inside the same jars used to grow the bacteria in test tubes. About 1 week of incubation was necessary before bacterial colonies were clearly visible on the agar plates. Strain MCB never grew on these plates; therefore, its purity was never certain. Similarly, the microorganisms of the mixed culture Ma grew very poorly on the agar plates, and attempts to isolate pure cultures of methanotrophs from this mixed culture were not successful.

The microorganisms were incubated at 30 °C on rotary shakers, but the analysis of the headspace of the experimental bottles was conducted at room temperature (about 24 °C).

Batch System

Almost all experiments were carried out in a batch system consisting of glass bottles (64 ml) (Qorpack clear Boston Rounds, Fisher Scientific, Rochester, NY) closed with screw caps with teflon-coated silicon septa or Mininert valves (Baxter Scientific Products, Edison, NJ). The bottles contained 8 mL of an aqueous solution of mineral salts. The microorganisms and the chlorinated compound were added to these bottles. TCE or TCA diluted in a small volume of water was added from a stock of TCE or TCA dissolved in water at a concentration equal or below 500 μg/mL.

Analytical Methods

TCE and TCA were determined with a gas chromatograph (GC) (Hewlett-Packard HP-5890A) fitted with a 25m HP-1 capillary column (crosslinked methyl silicone gum 0.2 mm x 0.33 μm film thickness), an electron-capture detector (ECD), and a split capillary inlet. All materials were from Hewlett Packard, Kennett Square, PA. The carrier gas was nitrogen; the headpressure of the column was 25 psi with a corresponding flow rate of about 1.74 mL/min, and the oven temperature was 55 °C. Under these conditions the retention time of TCA was about 1.76 min, and the retention time of TCE was about 2.08 min.

The glass insert of the split injector of the GC was deactivated to eliminate the tailing of the TCA and TCE peaks. Deactivation was accomplished by acid washing the insert overnight in concentrated sulfuric acid with Nochromix (an oxidizing agent manufactured by Godax Laboratories, New York, NY), soaking the insert for a few seconds in a 10-percent (vol/vol) solution of dichlorodimethylsilane in hexane, and rinsing it with acetone.

The first method used to analyze TCE and TCA in the experimental bottles involved the extraction of the chlorinated compound from the 8 ml of aqueous solution by adding 1.6 mL of hexane. Hexane was separated from the water and the microbial cells by centrifugation, and a 4-μL sample of this layer was analyzed by GC. The method, which was only used in preliminary experiments, had a detection limit of approximately 1 μg of TCE or TCA per liter of water. Pentane was initially used instead of hexane in this method, but its use was quickly abandoned because its high volatility led to large experimental errors.

The second method involved the analysis of the headspace of the experimental bottles, and it was used only for the analysis of TCE. A volume of 9 or 500 μL of the bottle headspace was removed with a gas-tight syringe and injected in the GC. The limit of detection of TCE in water was below 1 μg/L when the sampling volume was 9 μL, and close to

8

2 ng/L when the sampling volume was 500 μL. Because of its simplicity and sensitivity, this method was used in almost all the experiments.

At equilibrium, TCE will be present in the headspace of the bottle at a concentration proportional to its concentration in aqueous solution. The Henry constants H_T (ratio of the concentration of the test compound in the air (grams per liter) to its concentration in the liquid (grams per liter)) of TCE at different temperatures were obtained from Gossett (1987). These constants were used to calculate the concentration of TCE in the aqueous solution as a function of the concentration of TCE in the headspace of the experimental bottle when only the concentration in the headspace was determined with the GC.

Methane was analyzed by injecting a sample (9 μL) of the headspace of the experimental bottle into the GC. The column used was an HP-1 capillary column (diameter of 530 m and length of 5 m) and methane was determined with a flame ionization detector (FID). This column is not appropriate for separating gases, but methane was present in such large amounts in the bottles that its analysis was still accurate. A glass column (diameter of 2 mm and length of 6 ft) packed with Porapak Q 80/100 mesh was used for some analyses. Both columns were obtained from Hewlett Packard.

A method to detect chloride ion released during TCE oxidation was developed. This method "depends upon the displacement" by chloride ion "of thiocyanate ion from mercury (II) thiocyanate complex; in the presence of iron (III) ion a highly coloured iron (III) thiocyanate complex is formed, and the intensity of its color is proportional to the original chloride ion concentration" (Jeffery et al. 1989). The procedure given by Jeffery et al. (1989) was modified somewhat for convenience and to increase the detection limit for chloride. The procedure used consisted of adding to 1.0 mL of a saturated solution of $Hg(SCN)_2$ in ethanol 3.5 mL of the aqueous solution to be analyzed followed by the addition of 0.5 mL of 1.0 M $Fe(NO_3)_3 \cdot 9H_2O$ dissolved in an aqueous solution of 4.5 M HNO_3.

The detection limit was about 10 μM of chloride ion in solution. The solution to analyze was obtained from the aqueous solution of the experimental bottles by removing the microbial material first by centrifugation, then by filtration through a syringe filter (0.22 μm).

However, this method to detect chloride was only used once because some organic material remaining in the filtered solution apparently reacted with $Hg(SCN)_2$ similarly to chloride ion, and the result was an overestimate of the concentration of chloride in the solution. It is not known what organic compounds were responsible for this reaction, but the addition of cysteine (which has a -SH functional group) to a solution to be analyzed gave similar results as adding chloride ion to this solution.

3   Experiments

Degradation of TCE and TCA by Bacteria Growing on Different Organic Compounds

The purpose of the experiment was to investigate which bacterial enrichment was able to degrade TCE or TCA. The experimental bottles contained 8 mL of an aqueous solution of mineral salts and either 140 μg of TCE or 80 μg of TCA per liter of aqueous solution.

Four enrichments were used. They were grown on methane (enrichment Ma), propane, pentane, and hexane, respectively. The concentration of methane and propane in the air of the headspace of the experimental bottle was 20 and 40 percent, respectively. The concentration of pentane and hexane in the aqueous solution was 1.0 g/L.

For each microbial enrichment, six experimental bottles were used. The time at which TCE was added to the experimental bottle varied. TCE was added either together with the microbial inoculum or 1 to 5 days later. The size of the microbial inoculum was 40 μL. Ten days after the beginning of the experiment, the aqueous solution was analyzed for its content of TCE or TCA after extraction with hexane.

All the microorganisms grew readily under the experimental conditions, except that the culture utilizing propane grew slowly. The enrichment growing on methane markedly reduced the concentration of TCE, the value falling to approximately 1 μg/L in water. The propane degraders decreased the concentration of TCE by up to 50 percent, but significant degradation of TCE by the pentane and hexane degraders was not observed. None of the enrichments were able to appreciably reduce the concentration of TCA. However, the error associated with the analytical method was more than 10 percent so that slight activity on either chlorinated compound would not have been detected.

A more extensive degradation of TCE occurred in experimental bottles in which TCE was added less than 3 days after the inoculum of methane oxidizers, or when TCE was added together with propane oxidizers.

10

Based on these results, only the biodegradation of TCE by methane- and propane-oxidizing bacteria was studied.

Toxicity Level of TCE to Methane and Propane Oxidizers

An experiment was conducted to determine the lowest aqueous concentration of TCE that inhibits microbial growth. In addition, TCE was added to the solution at different times following the addition of 40 μL of the inoculum.

The inocula consisted of three mixed cultures (Ma, Mb, and Mc) and two pure cultures (MCB and Mx) of methane oxidizers and a culture of propane oxidizers. The headspace of the experimental bottles contained either 45-percent methane in the air or 45-percent propane in the air. The concentrations of TCE in the aqueous phase of the bottles were 0.12, 1.4, 8.4, or 28.0 mg/L.

When the concentration of TCE was 0.12 mg/L, four experimental bottles were used for each inoculum, but TCE was added at different times. When the carbon source was methane, TCE was added at the time of inoculation or 1, 2, or 4 days later. When the carbon source was propane, TCE was added at the time of inoculation or 4 or 9 days later (two bottles were inoculated at 9 days).

When the concentration of TCE was 1.4, 8.4, and 28.0 mg/L, only two experimental bottles were used for each inoculum and each TCE concentration. In the first bottle TCE was added to the microorganisms. TCE was added to the second bottle 4 days later when the bottle contained methane oxidizers or 10 days later when the bottle contained propane oxidizers. In addition, analyses were conducted of the contents of sterile bottles corresponding to each treatment to detect abiotic losses of TCE. The duration of the experiment was 12 days for methane degraders and 16 days for propane degraders.

Many measurements were made of the disappearance of TCE and methane from the different bottles. However, appreciable abiotic losses of TCE from the experimental bottles were detected. For this reason, the data are not presented. One reason for the abiotic losses of TCE was the damage to the teflon layer of the septa that closed the bottles. As a consequence, these damaged septa permitted TCE to reach and sorb to the silicone side of the septum. Nevertheless, the following observations were made.

When its concentration in the aqueous phase was 0.12 mg/L, TCE was degraded by all five inocula of methane degraders, and its final concentration sometimes was below 1 μg/L. Less extensive degradation of TCE

usually occurred when TCE was added after 4 days. In the presence of propane, degradation of TCE was slower than in the presence of methane, presumably because the bacteria grew more slowly; nevertheless, the TCE concentration fell to less than 1 µg/mL.

When the concentration of TCE was 1.4, 8.4, and 28.0 mg/L, its degradation could not be detected because of appreciable losses of TCE from the experimental bottles. However, observations of the turbidity in the bottles indicated that all the methanotrophic bacteria grew in the presence of 1.4 or 8.4 mg of TCE per liter, but none could grow in the presence of 28.0 mg of TCE per liter. Therefore, the toxic level of TCE in an aqueous solution for the methane oxidizers was between 8.4 and 28.0 mg/L.

Propane degraders were not able to grow in the presence of 1.4, 8.4, and 28.0 mg of TCE per liter. Growth was not detected after 19 days at these concentrations, whereas analysis of bottles without TCE indicated that, in the absence of TCE, these bacteria would give a turbid solution after about 6 days. Apparently, the toxic level of TCE in aqueous solution for the mixed culture of propane oxidizers was between 0.12 and 1.4 mg/L.

TCE Sorption on Dry GAC

Air streams contaminated with TCE may contain 10 mg/kg of TCE. At equilibrium with water and at room temperature, this represents somewhat more than 30 g of TCE per liter in the aqueous phase. At this low concentration and in a bioreactor that is flushed with large volumes of air, a single-stage bioreactor might function far below its TCE-degrading capacity and be prone to drying. Therefore, this and the following experiments were devised to investigate whether TCE from contaminated air could be trapped by activated carbon and then degraded by methane- or propane-oxidizer bacteria.

Three kinds of activated carbon were used: granular activated carbon (GAC) (mesh 6-14, Fisher Scientific Co, Rochester, NY), the same activated carbon but crushed to a powder, and a fine activated carbon (grade G-60, Atlas Chemical Industries, Inc., Chemical Division, Wilmington, DE).

This experiment was devised to investigate the sorptive capacity of GAC towards TCE. The experiment consisted of adding different amounts of TCE to duplicate experimental bottles containing 500 mg of GAC and measuring the concentration of TCE in the headspace of the bottle at different times until equilibrium was reached.

Approximately 6 days was necessary for equilibrium to be reached between TCE sorbed to GAC and TCE in the headspace. The results obtained after 1 and 6 days are shown in Figure 2. The data indicate that

12
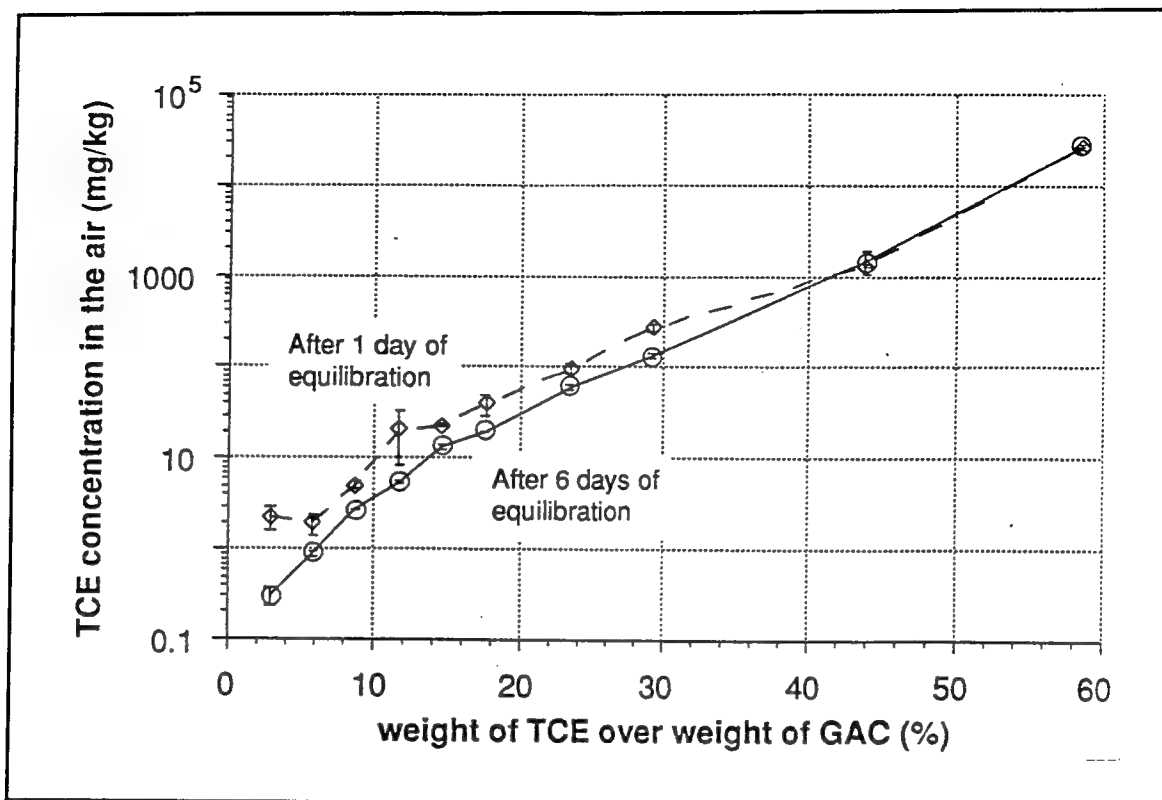
Figure 2. Concentration of TCE in the air in equilibrium with different amounts of TCE sorbed to GAC

10 mg of TCE per kilogram in the gas phase will be in equilibrium with the equivalent of 14 g of TCE sorbed on 100 g of initially uncontaminated GAC, or 14 percent. The data also show that after 1 day, TCE was still not in complete equilibrium with the GAC, especially with low amounts of TCE.

TCE Sorption on Wet GAC

This experiment was carried out to determine affinities for TCE of three kinds of dry or wet activated carbon. The experiment consisted of adding 6 µL of TCE (8.8 mg TCE) to experimental bottles containing 80 mg of three kinds of activated carbon: GAC, crushed GAC, and fine activated carbon. The concentration of TCE in the headspace of the bottle was measured a few minutes following the addition of TCE and 3 and 4 days thereafter. On day 4, 8 mL of water was introduced into the experimental bottles so that the activated carbon was completely submerged, and the concentration of TCE in the bottle headspace was measured a few minutes thereafter and 1 day later (day 5).

The results (Figure 3) indicate that GAC (crushed or not) apparently had a greater affinity for TCE than the fine activated carbon, as only 8 mg of TCE were present per kilogram of headspace in the bottles containing GAC and crushed GAC, whereas about 230 mg of TCE were present per kilogram of headspace in the bottles containing fine activated carbon. In addition, an appreciable amount of TCE was released to the headspace following the addition of water. This release was apparently faster with crushed GAC and fine activated carbon. After 1 day, the concentration of TCE in the headspace of the experimental bottles was about 330 mg/kg in the presence of GAC, 650 mg/kg in the presence of crushed GAC, and 8,300 mg/kg in the presence of fine activated carbon.

This release of TCE following the addition of water may facilitate the biological degradation of TCE sorbed on dry GAC by decreasing the ability of GAC to sorb TCE. In addition, the results show that if GAC is used as a solid packing in a wet bioreactor, its sorptive capacity for TCE will be substantially lower than its sorptive capacity in a dry environment. In addition, if a column of dry GAC is used to trap TCE from a contaminated air stream, its sorptive capacity is likely to vary depending on the water content of the air stream (Crittenden et al. 1988).
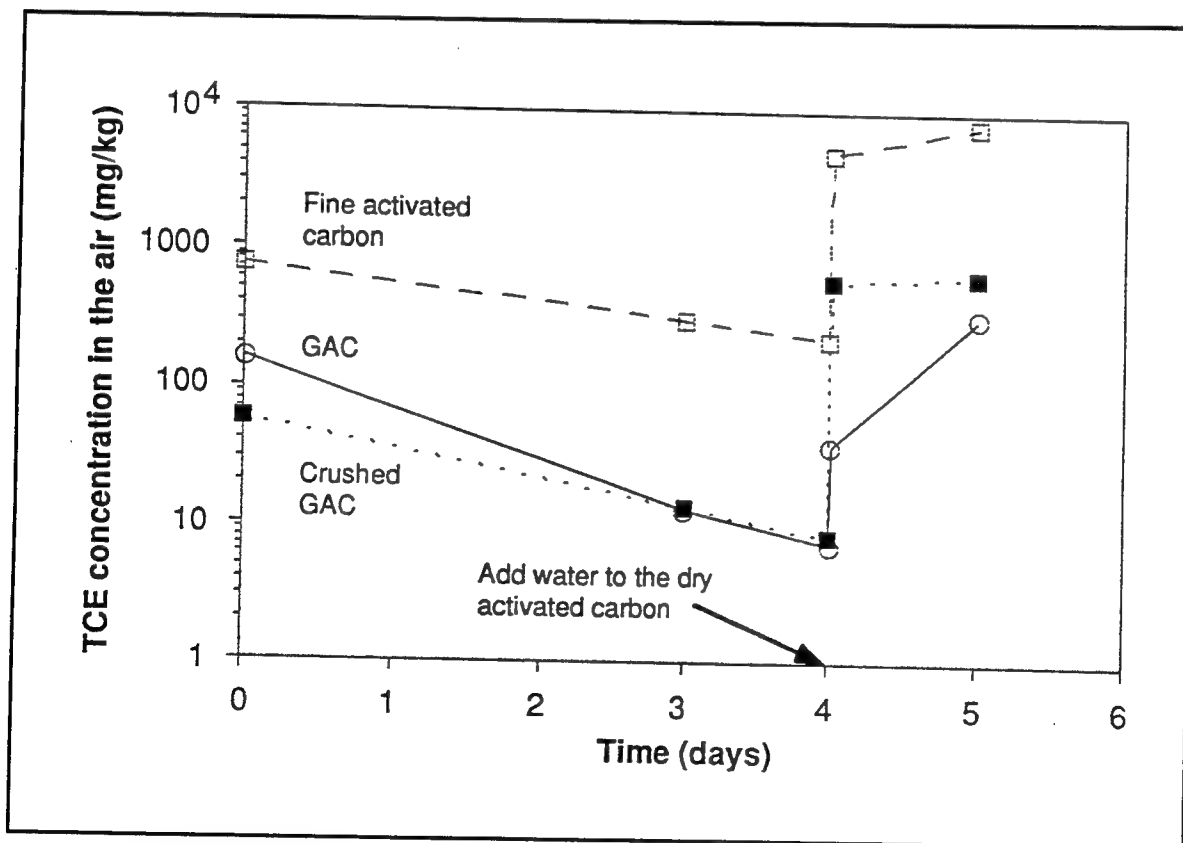
Figure 3. Sorption of 8.8 mg of TCE to 80 mg of dry and wet activated carbon. The activated carbon was initially dry

Degradation of TCE Sorbed to Activated Carbon

An experiment was conducted to study the biological degradation of TCE sorbed on GAC. The first reason for the study was that GAC could be used as packing material in a bioreactor designed to decontaminate an air stream contaminated with low concentrations of TCE. The TCE in the airstream would be expected to be strongly sorbed to the GAC before the bacteria begin to degrade it (unless the GAC is already saturated with TCE). Therefore, it is important to determine whether TCE sorbed to GAC could be effectively degraded by bacteria or if it is not available for bacterial uptake.

The second reason is that dry GAC columns could be used to trap TCE from a contaminated air stream, and then the contaminated GAC of the columns could be treated by a microbiological process. This procedure would avoid problems associated with flushing a bioreactor with huge amounts of contaminated air, because TCE would no longer be present in the air stream but would be sorbed to GAC. In addition, as mentioned above, some of the sorbed TCE would be released when the GAC is placed in the aqueous solution in the bioreactor, and TCE might thereby become more available to bacteria.

Several bottles containing 8 mL of an aqueous solution of mineral salts and 8 mg of TCE sorbed to 80 mg of the three kinds of activated carbon used in the previous experiment were inoculated with methane- and propane-degrading microorganisms. Because GAC has a large sorptive capacity for TCE, it was expected that only a small fraction of the TCE would be degraded. For this reason, measuring the amount of TCE remaining in the solution was not considered to be a sufficiently sensitive method. Instead, it was planned to estimate TCE degradation by measuring the amount of chloride released to the aqueous solution following the oxidation of TCE.

Unfortunately, organic constituents of the bacteria as well as substances in the activated carbons also reacted with the chloride reagent ($Hg(SCN)_2$) so that analyses could not be performed by such procedures.

Extraction of TCE from GAC with Methanol

A method was investigated to remove TCE from GAC. It consists of extracting the TCE with methanol. The TCE dissolved in methanol would then presumably be degraded by methanotrophs. An experiment was thus conducted to estimate how much TCE could be extracted from GAC by repetitive extraction with small volumes of methanol.

A volume of 40 μL TCE (59 mg) was added to two duplicate experimental bottles containing 500 mg of GAC, and the bottles were allowed to stand for 1 week to permit equilibrium to be reached between TCE in the headspace and TCE sorbed to GAC. Then 1 mL of methanol was added to the bottles, and after 1 hr and repetitive shaking, the methanol was removed from the bottles, and its content of TCE was measured. This procedure was repeated several times, although sometimes the methanol was removed after 1 day.

The results in Figure 4 show that after six extractions with 6 mL of methanol, about 75 to 80 percent of the TCE could be extracted from the GAC. The first time methanol was added into the bottles, only about 0.4 mL could be removed, the remainder being sorbed to the GAC. In the extractions that followed the first one, usually all of the added methanol could be recovered. The concentration of TCE was 5.2 μL of TCE per milliliter of methanol or 0.0096 g of TCE per gram of methanol.

Although it is difficult to extrapolate these results to conditions in which thorough extraction of TCE from GAC was achieved, the results suggest the concentration of TCE that might be obtained in a solution of methanol used for the extraction. This methanol can then be provided to methanotrophs in the expectation that they will degrade TCE while growing on methanol. However, it was necessary first to investigate the
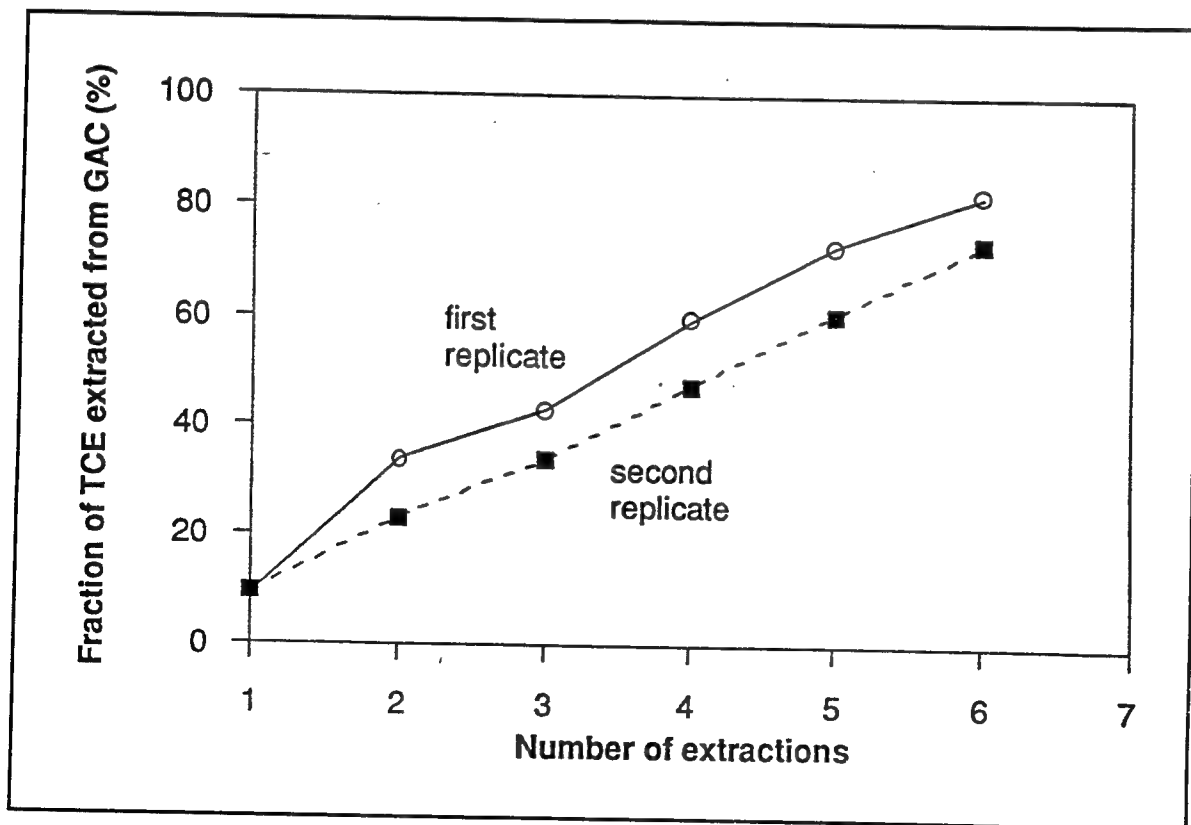
Figure 4. Extraction with methanol of 59 mg TCE sorbed to 500 mg GAC

toxicity of methanol to methanotrophic bacteria. This is the subject of the next experiment.

Methanol Toxicity to Methanotrophic Bacteria

The concentration of methanol above which the growth of methanotrophic bacteria on methanol will be adversely affected was assessed. Mixed cultures Ma, Mb, and Mc and pure cultures MCB, Mx, M1, M2, and M3 were grown in bottles containing 8 mL of an aqueous solution of mineral salts with 0.1, 0.5, 1.0, 2.0, and 5.0 percent (vol/vol) of methanol as the sole source of carbon and energy.

A qualitative estimation of the growth of the bacteria was obtained by visually observing the turbidity of the aqueous solution in each bottle 1, 3, and 6 days after the addition of 40 µL of the inocula. The extent of turbidity was designated 0 (no turbidity) or by an increasing number of plusses. After 1 day, all the mixed cultures (Ma, Mb, and Mc) had grown to their maximum turbidity when the concentration of methanol was up to 0.5 or 1.0 percent, and less turbidity was observed at 2.0- or 5.0-percent methanol (Table 1). The pure cultures grew slower than the mixed cultures, and no growth was detected at 5.0-percent methanol.

Table 1
Methanol Toxicity. Turbidity in Bottles After 1 Day of Incubation

| Inoculum | Methanol Concentration in Solution, percent | | | | |
	0.1	0.5	1.0	2.0	5.0
Ma	++++	++++	++++	+	+
Mb	++++	++++	+++	+	+
Mc	++++	++++	+++	+	+
MCB	+	+	+	+	0
Mx	+	+	+	+	0
M1	+	+	+	+	+
M2	0	0	0	0	0
M3	+	+	+	+	0

After 3 days, differences in turbidity were not observed at any concentration of methanol with mixed cultures Ma and Mc (Table 2); culture Mb was only somewhat less turbid at 5.0-percent methanol than at lower concentrations. With the exception of strain M1, no pure culture could grow at 5.0-percent methanol. In addition, with strains Mx, M1, and M3 the

aqueous solution was less turbid at 2.0-percent methanol than at lower concentrations.

Table 2
Methanol Toxicity. Turbidity in Bottles After 3 Days of Incubation

Inoculum	Methanol Concentration in Solution, percent				
	0.1	0.5	1.0	2.0	5.0
Ma	++++	++++	++++	++++	++++
Mb	++++	++++	++++	++++	+
Mc	++++	++++	++++	++++	++++
MCB	+	+	+	+	0
Mx	+++	+++	+	+	0
M1	+	+	+	+	+
M2	+	+	+	+	0
M3	+	+	+	+	0

After 6 days, differences in turbidity were not evident at any concentration of methanol in the bottles containing the mixed cultures Ma, Mb, and Mc (Table 3). Pure cultures MCB, M1, and M3 showed a little turbidity at 5.0-percent methanol, but strains Mx and M2 did not grow at that concentration. The sensitivity of the pure cultures to methanol varied, and they did not reach the turbidity obtained with mixed cultures, even at the lowest concentration of methanol (0.1 percent).

Table 3
Methanol Toxicity. Turbidity in Bottles After 6 Days of Incubation

Inoculum	Methanol Concentration in Solution, percent				
	0.1	0.5	1.0	2.0	5.0
Ma	++++	++++	++++	++++	++++
Mb	++++	++++	++++	++++	++++
Mc	++++	++++	++++	++++	++++
MCB	+	+	+	+	+
Mx	+++	+++	+	+	0
M1	+	+	+	+	+ .
M2	+	+	+	+	0
M3	+	+	+	+	+

Apparently, the pure cultures were more sensitive to methanol than the mixed cultures. This is not surprising considering that the mixed cultures are likely to contain some methylotrophic bacteria that grow efficiently on methanol, whereas methanotrophic bacteria (like the purified strains) grow slowly on methanol (Anthony 1986). The populations of the mixed cultures were probably composed mainly of methylotrophic bacteria, whereas the mixed cultures probably contain mainly methanotrophic bacteria when grown in the presence of methane. The concentration of methanol above which growth of the purified strains was substantially affected was apparently between 2.0 and 5.0 percent, but an effect on growth was even evident below these concentrations.

Degradation of TCE by Methanotrophs Growing on Methanol

An experiment was conducted to investigate whether methanotrophs would degrade TCE while growing on methanol. As mentioned above, the purpose of this experiment was to test if methanotrophs could be used to degrade a solution of methanol contaminated with TCE as obtained following the extraction with methanol of TCE sorbed to GAC.

The microbial inocula consisted of three mixed cultures (Ma, Mb, and Mc) and four pure cultures (MCB, Mx, M1, and M3). The inocula were grown in bottles containing 8 mL of an aqueous solution of mineral salts and about 250 g of TCE per liter of water. The concentration of methanol was 0.8 percent, which was not toxic to the organisms (see previous experiment). Some bottles also contained a small amount of methane (0.71 or 0.016 percent in the air of the headspace). One of the purposes of the addition of methane was to test whether the presence of small amounts of methane would influence the degradation of TCE. The second purpose was to study the biodegradation of methane by methanotrophs growing on high concentrations of methanol. For this study, the degradation of methane was only measured in the experimental bottles containing 0.71 percent of methane. Methane was introduced into the headspace of the bottle, either together with the inoculum or when the aqueous solution became turbid as a result of microbial growth. A control was also tested in which 0.71 percent of methane but no methanol was present.

All tests were conducted in triplicate, and eight sterile bottles were used as controls. The system was incubated for 15 days, and the contents of the bottles were analyzed six times during this period for TCE and sometimes methane and methanol.

A total of 128 bottles were used in this experiment. Because TCE was not appreciably degraded in any of the bottles, only a few of the analytical results are presented. The percentage of TCE and methane remaining after 15 days in the bottles inoculated with microorganisms Ma, Mb, and Mc

and also with microogranisms MCB, Mx, M1, and M2 is shown in Figures 5 and 6. An appreciable disappearance of TCE was not evident in most cultures containing methanol. In the few instances in which the concentration of TCE had decreased, the maximum loss of TCE as a result of microbial activity was never more than about 60 percent of the amount added. An average of 25 percent of the TCE disappeared abiotically from the control bottles, indicating that similar losses of TCE may have occurred in the experimental bottles. Thus, most of the TCE missing from the experimental bottles may have disappeared abiotically and not as a result of bacterial activity.

From 70 to 90 percent of the methane (initially present at 0.71-percent methane in air) remained in experimental bottles that contained methanol, but only about 3 percent of the methane remained in experimental bottles in which methanol was absent. The only exception is one of the triplicate experimental bottles that was inoculated with strain Mx, in which 62 percent of the methane remained at the end of the experiment. The abiotic loss of methane from the experimental bottles was not assessed, but it may be responsible for the small amount of methane lost in the bottles containing methanol.

Most experimental bottles containing methanol were turbid due to microbial growth after 2 days. The bottles containing 0.71-percent methane and no methanol were only slightly turbid, and they contained a much lower density of microbial cells than the bottles containing methanol. Bottles initially containing 0.016 percent of methane and no methanol did not show visible growth, and significant TCE disappearance was not evident, probably because the cell density was too low.

In all bottles containing methanol, methanol was still present at high concentrations (about 0.5 percent in water) at the end of the test period (data not shown). It is likely that the bacteria had used most of the oxygen in the headspace and consequently could no longer oxidize methanol.

The previous experiment on the toxicity of TCE to methanotrophs indicated that the inocula Ma, Mb, Mc, MCB, and Mx were able to degrade TCE while growing on methane. A preliminary test experiment (data not shown) indicated that strains M1 and M3 were also able to degrade TCE while growing on methane. However, in a larger experiment dealing with the possible existence of a threshold for TCE degradation (an experiment that was made before the one described in this section), only inocula Ma, Mb, MCB, M2, and M3 were able to degrade TCE while growing on methane, whereas inocula Mc, Mx, and M1 were not able to degrade TCE while growing on methane. In addition, an experiment conducted after the one described in this section indicated that, in the presence of methane, strains MCB, M3, and M4 but not strains Mx, M1, and M5 could degrade TCE.

20
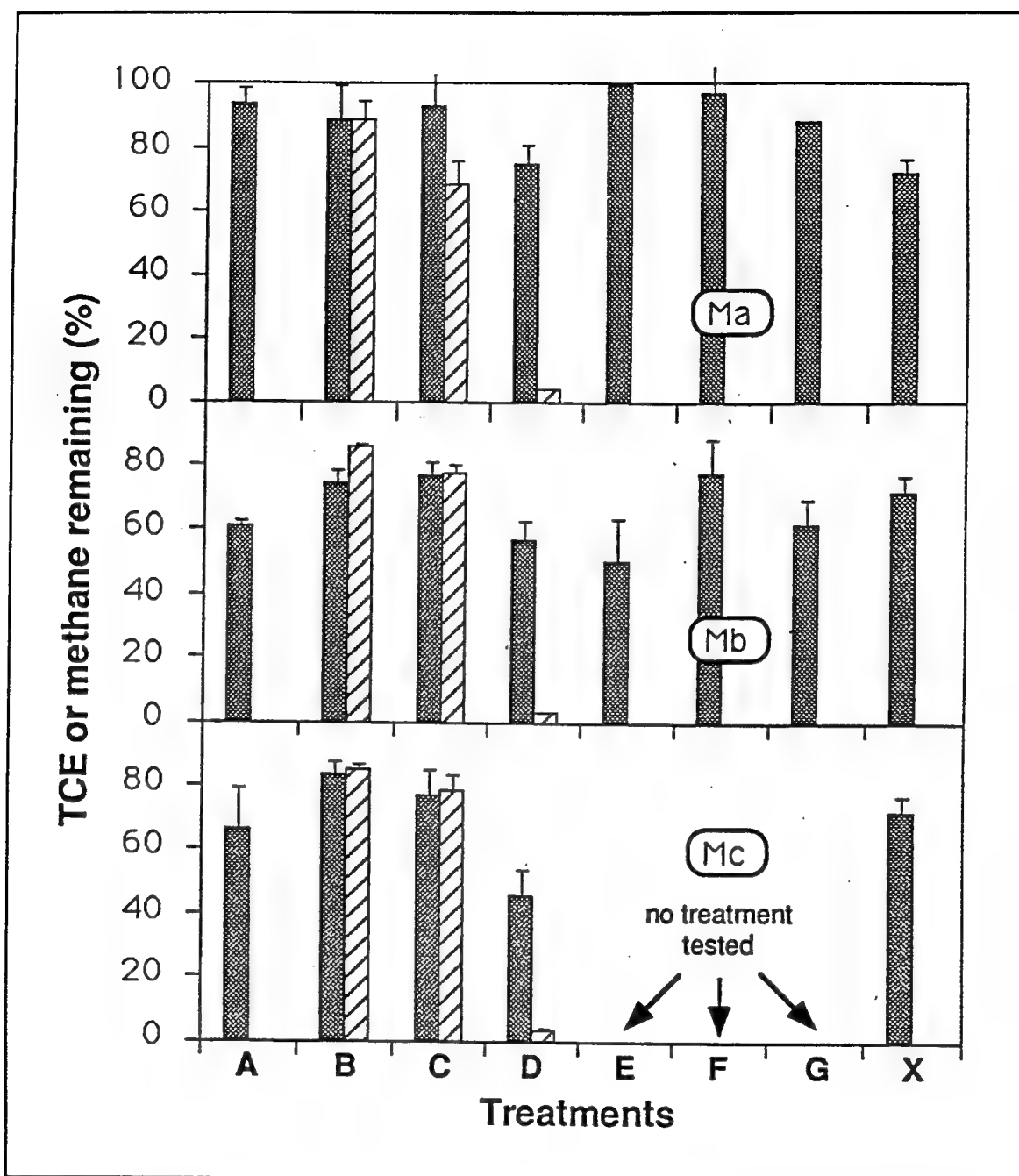
Figure 5. Degradation of TCE (250 µg/l) and methane by methanotrophs (Ma, Mb, Mc) growing on 0.8-percent methanol (MeOH) and/or methane (0.71 percent or 0.016 percent in air). The columns indicate the fraction (percent) of TCE and methane remaining in the experimental bottles after 15 days. Treatments: A, MeOH; B, MeOH + 0.71-percent methane that was added when the solution was turbid; C, MeOH + 0.71-percent methane that was added initially; D, 0.71-percent methane (no MeOH); E, like B but with 0.016-percent methane, F, like C but with 0.016-percent methane; G, 0.016-percent methane (no MeOH); X, sterile control. TCE: ▓ ; Methane: ▨ (only the experimental bottles initially containing 0.71 percent of methane were analyzed)
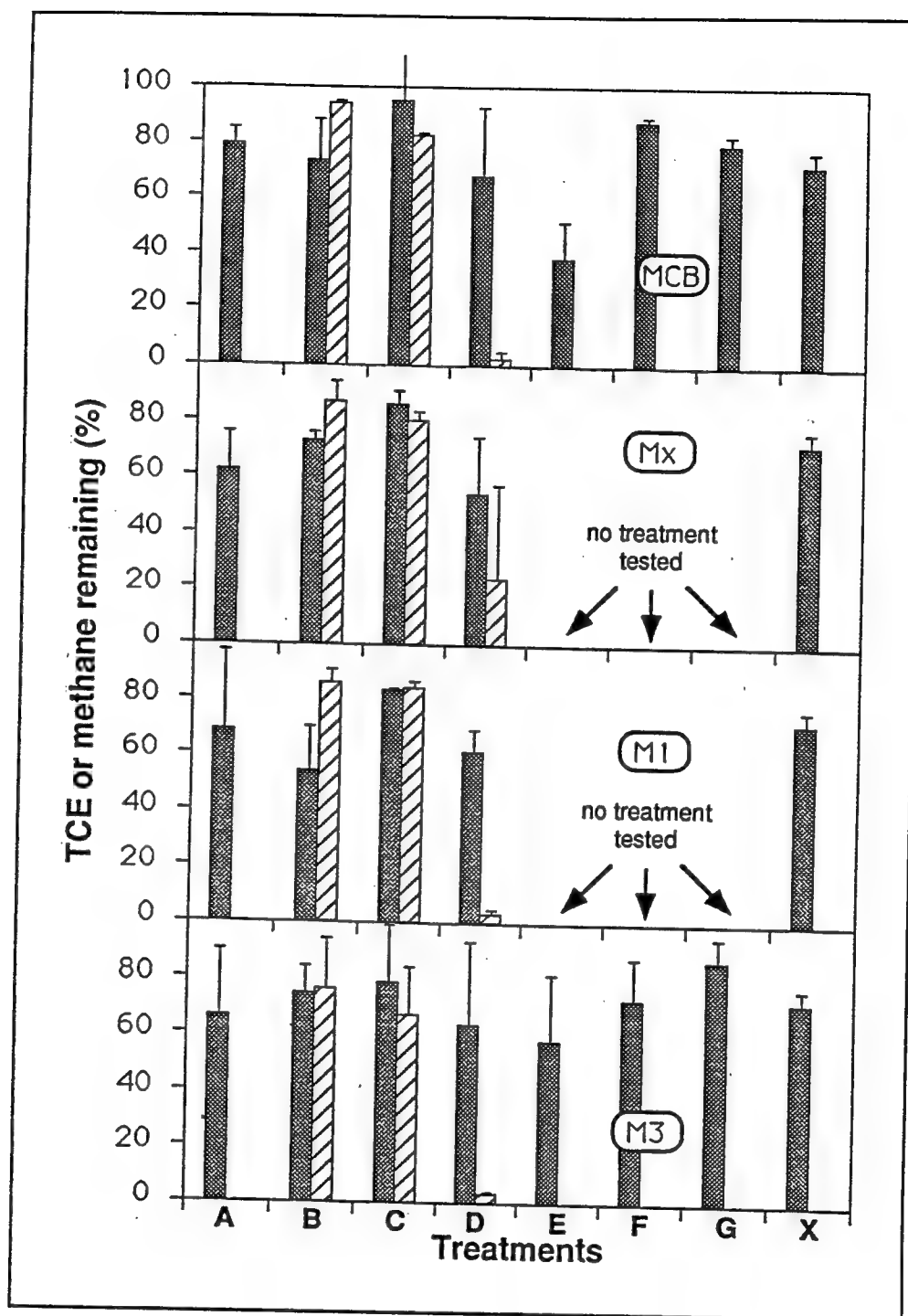
Figure 6. Degradation of TCE (250 μg/L) and methane by methanotrophs (MCB, Mx, M1, M3) growing on 0.8-percent methanol (MeOH) and/or methane (0.71 percent or 0.016 percent in air). See the definition of the treatments in Figure 5. TCE: ▨ ; methane: ▧

Consequently, it seems that microorganisms Mc, Mx, and M1 lost their ability to degrade TCE while growing on methane. This is surprising because strain M3, which can degrade TCE, was isolated from the mixed culture Mc. The reason for this loss of ability to degrade TCE is unknown.

Thus, the apparent inability of some inocula (especially inocula Mx, Mc, and M1) to degrade TCE while growing on methanol may simply be due to the inability of these inocula to degrade TCE under any conditions.

It is known that the enzyme responsible for the primary oxidation of TCE is sMMO, which also catalizes the oxidation of methane to methanol. It is likely that the synthesis and expression of sMMO is tightly controlled by the cell to prevent unnecessary oxidation of methane to methanol which would lead to accumulation of methanol in the cell. One of these physiological controls may be sensitive to the concentration of methanol, and a high concentration of methanol in the aqueous medium may inhibit the synthesis of sMMO (or turned off the enzyme by some allosteric mechanism). Another explanation for the results is that sMMO has a weak capacity to oxidize methanol to formaldehyde, and methanol will therefore act as a competitive inhibitor of methane for sMMO, an effect that is increased by the high concentration of methanol.

In summary, the data suggest that the degradation of TCE and methane by methanotrophs is strongly if not completely inhibited in the presence of high concentrations of methanol. Therefore, the use of high concentrations of methanol as a growth substrate for methanotrophs to cometabolize TCE in a bioreactor does not appear to be a feasible means to cometabolize TCE.

Influence of Packing Material on the Cometabolism of TCE

In a bioreactor designed to carry out the degradation of TCE by methanotrophic bacteria, the identity of the packing material may influence the degradation process of TCE. Therefore, an experiment was conducted to study the degradation of TCE in bottles containing an aqueous solution of mineral salts, methanotrophic bacteria, TCE, and different possible packing materials for a bioreactor.

The materials were glass beads, ceramic saddles (6 mm) (both from Fisher Scientific Co, Rochester, NY), marble chips, untreated sand, combusted sand, and a material probably made of ceramic and clay particles used for cat litter. These materials were thoroughly washed with water before use, and a portion of each was kept several hours in a boiling aqueous solution of Na_2EDTA (2 g/L), after which they were again washed extensively. The EDTA was used to remove any copper ions that might have

been present, as most methanotrophs are unable to cometabolize TCE when grown in the presence of substantial copper concentrations.

The microbial inocula consisted of two enrichments (Ma and Mb) and one inoculum containing two pure cultures (M3 and M4). All inocula had the ability to cometabolize TCE when grown on methane. From 3 to 8 g of packing material was added to the bottles and these were supplemented with 4 mL of an aqueous solution of mineral salts. The inoculum and a mixture of methane and air (about 20-percent methane) was introduced. The bottles were incubated at 30 °C for 5 days, at which time the bottles were turbid from microbial growth. The bottles were then flushed with a fresh mixture of methane and air (about 20-percent methane), and TCE was added at a concentration of about 80 µg/L in water. Analyses for methane and TCE were performed after 2 and 4 days. All tests were conducted in triplicate.

The results in Figure 7 (no packing material, glass beads, and ceramic saddles) and Figure 8 (marble chips, combusted sand, untreated sand, and cat litter) show the concentrations of TCE remaining in water 2 and 4 days after the addition of TCE to the bottles. None of the packing materials tested sorbed significant amounts of TCE, as indicated by the absence of large differences in TCE concentrations obtained in sterile condition in the presence and absence of packing material. With all the packings treated with EDTA and with glass beads and ceramic saddles not treated with EDTA, more than 80 percent of the TCE had disappeared with all microbial inocula 2 days after the addition of TCE. A somewhat smaller (60 percent or more) amount of TCE was metabolized by two inocula (Mb and M3/M4) in the presence of marble chips, untreated and combusted sand, and cat litter that were not treated with EDTA. However, the third culture (Ma) had almost no activity on untreated and combusted sand and cat litter that were not treated with EDTA, and it had less activity on marble chips untreated with EDTA than the two other inocula.

These results indicate that some packings can indeed inhibit TCE degradation by some methanotrophs. However, this inhibition is overcome if the packing is treated with EDTA. It is assumed that the differences observed between the treatments were the consequence of the higher concentration of copper ions in the packings not treated with EDTA. The reason why one inoculum (Ma) was more sensitive to the treatment with EDTA than the other inocula is not known and was not investigated.

Threshold in Cometabolism of TCE by Methanotrophs

The purpose of these experiments was to determine whether there is a threshold for the biodegradation of volatile organic compounds cometabolized and degraded as a consequence of microbial growth. In this

24
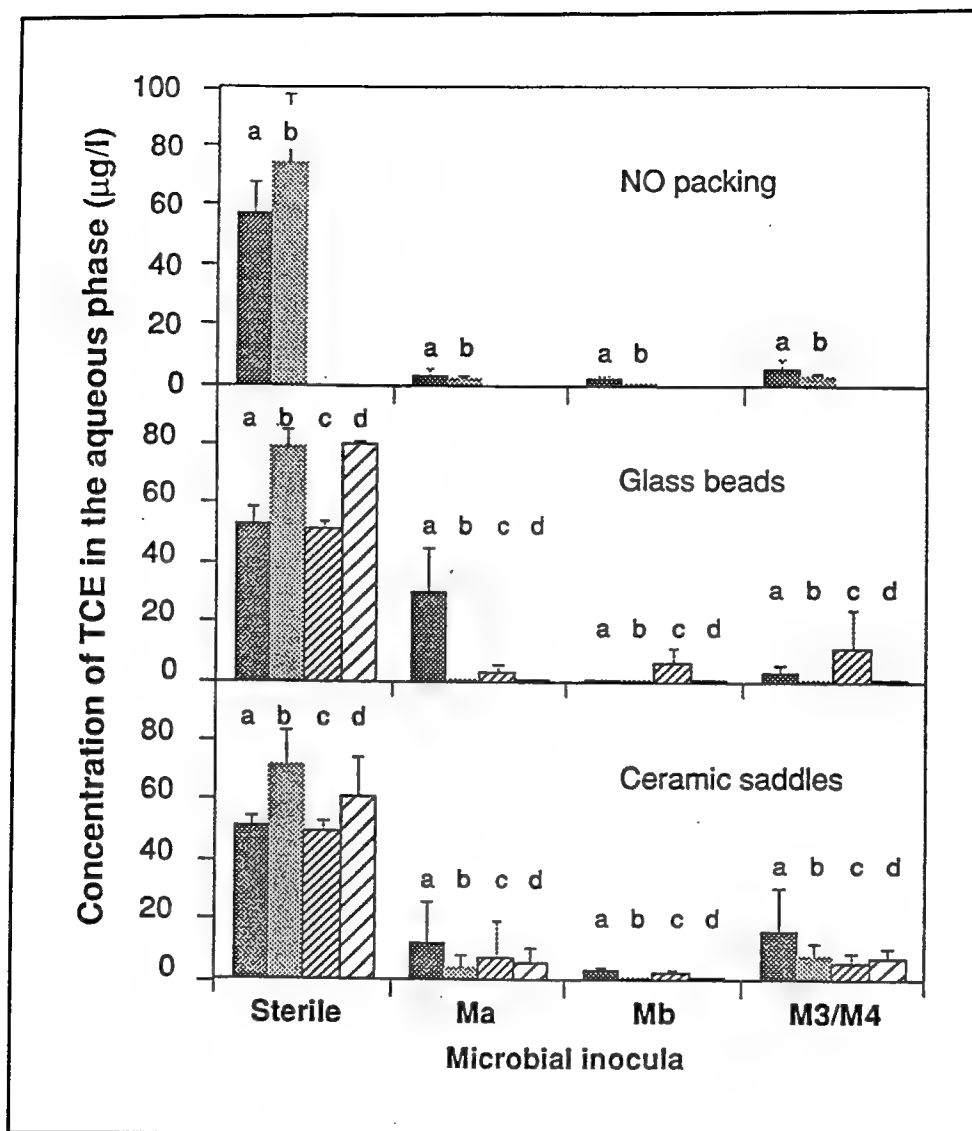
Figure 7. Degradation of 80 µg/L of TCE by methanotrophs in the presence of packing materials: no packing, glass beads, and ceramic saddles. The columns indicate the concentration of TCE remaining in the aqueous phase 2 and 4 days after the addition of TCE in the experimental bottles. Treatments: a—packing not treated with EDTA, day 2; b—packing not treated with EDTA, day 4; c—packing treated with EDTA, day 2; d—packing treated with EDTA, day 4
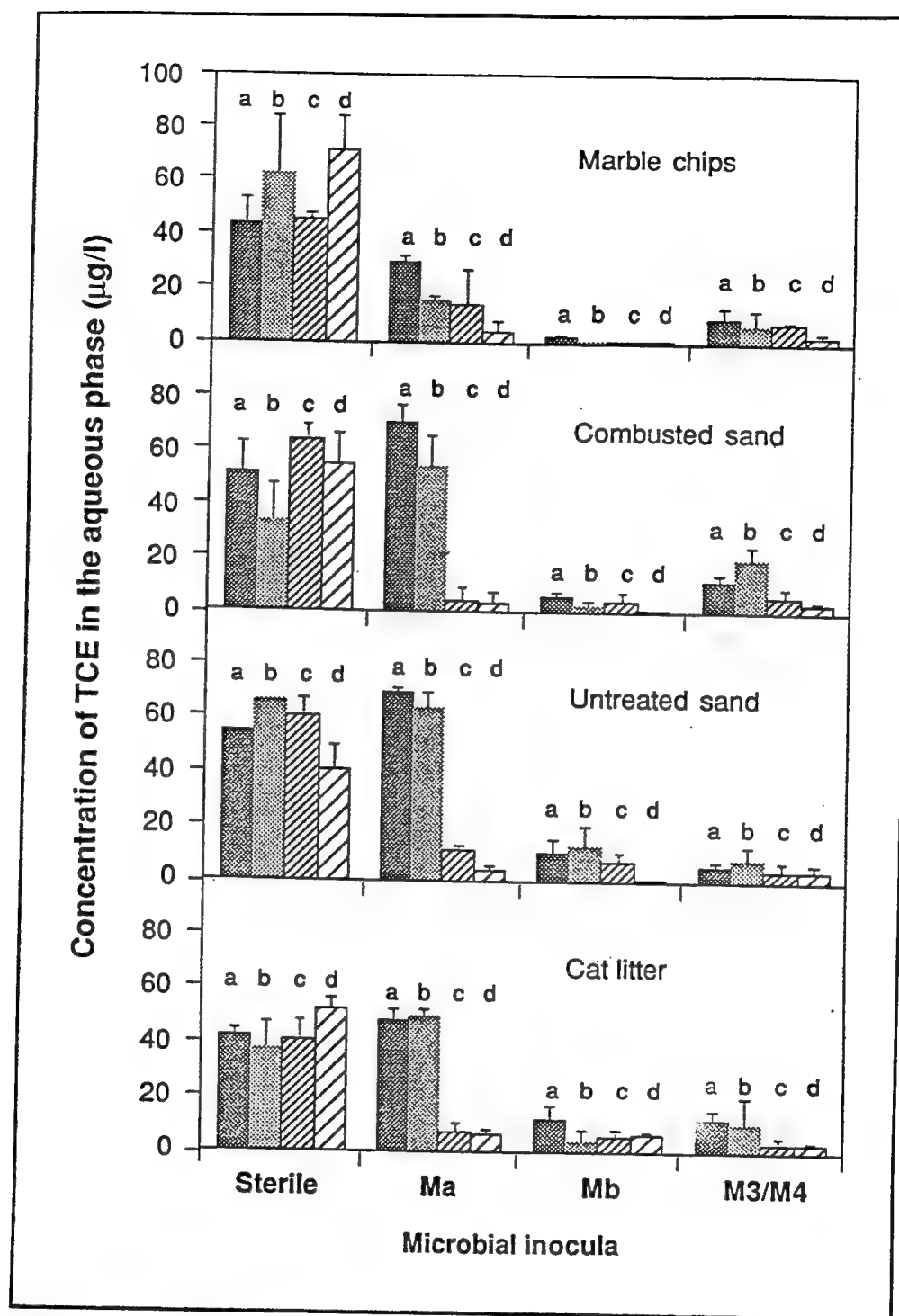
Figure 8. Degradation of 80 µg/L of TCE by methanotrophs in the presence of packing materials: marble chips, combusted sand, untreated sand, and cat litter. The different columns (a, b, c, d) are defined in Figure 7

particular case, the test compound was TCE, and the microorganisms responsible for its cometabolic degradation were all methanotrophs.

The setup of the experiments consisted of bottles containing 20 to 30 percent of methane in air and 8 mL of an aqueous solution of inorganic salts. TCE was added to the bottles so that its concentration in the aqueous solution was either 100 or 1 μg/L. The inocula consisted of mixed cultures Ma, Mb, and Mc, and pure cultures MCB, Mx, M1, M2, and M3. A small volume of inoculum (40 μL) was added to each bottle, except for some bottles which remained sterile and were used to estimate the abiotic losses of TCE during the experiments and the sampling process. The experiments were conducted using four replicates for each bottle. Sampling was conducted daily during the first week and then at longer time intervals for the following week, and the frequency varied depending on the type of microorganism used. TCE was analyzed by headspace sampling by removing either 9 or 500 μL of the bottle headspace, depending on the concentration of TCE remaining in the experimental bottle. The concentration of methane in the headspace was also analyzed, although the analyses gave little useful information.

A summary of the results is given in Figure 9. When acted on by microorganisms Mb, MCB, M2, and M3, the concentration of TCE at the end of the experiments (in at least 2 of the 4 replicates) was below the detection limit of the analytical method (about 0.002 μg of TCE per liter of water). This occurred when the starting concentration of TCE was either 100 or 1 μg/L. In enrichment Ma, TCE was still detected at about 0.020 μg/L in all bottles. Microorganisms Mc, Mx, and M1 did not degrade TCE appreciably at either the 100- or 1-μg/L concentration; these findings were surprising as Mc and Mx were apparently active on TCE in previous studies.

The data in Figure 9 also show that the rates of TCE degradation varied among the inocula. The amount of TCE degraded after 4 days was more than 90 percent for inoculum Mb, about 80 percent for M2 and M3, 70 percent for Ma, and only 30 percent for MCB. The values of all four replicates agreed in all experiments, even though some of the replicates were sometimes losing some TCE abiotically because a few of the bottles were closed by a damaged septum.

In summary, when the microorganisms were able to degrade TCE, they degraded it either below the detection limit (inocula Mb, MCB, M2, and M3) or to about 0.020 μg/L (inoculum Ma). Consequently, if there is a threshold for TCE cometabolism by methanotrophs, it is at concentrations below about 0.002 μg/L of TCE in the aqueous phase.
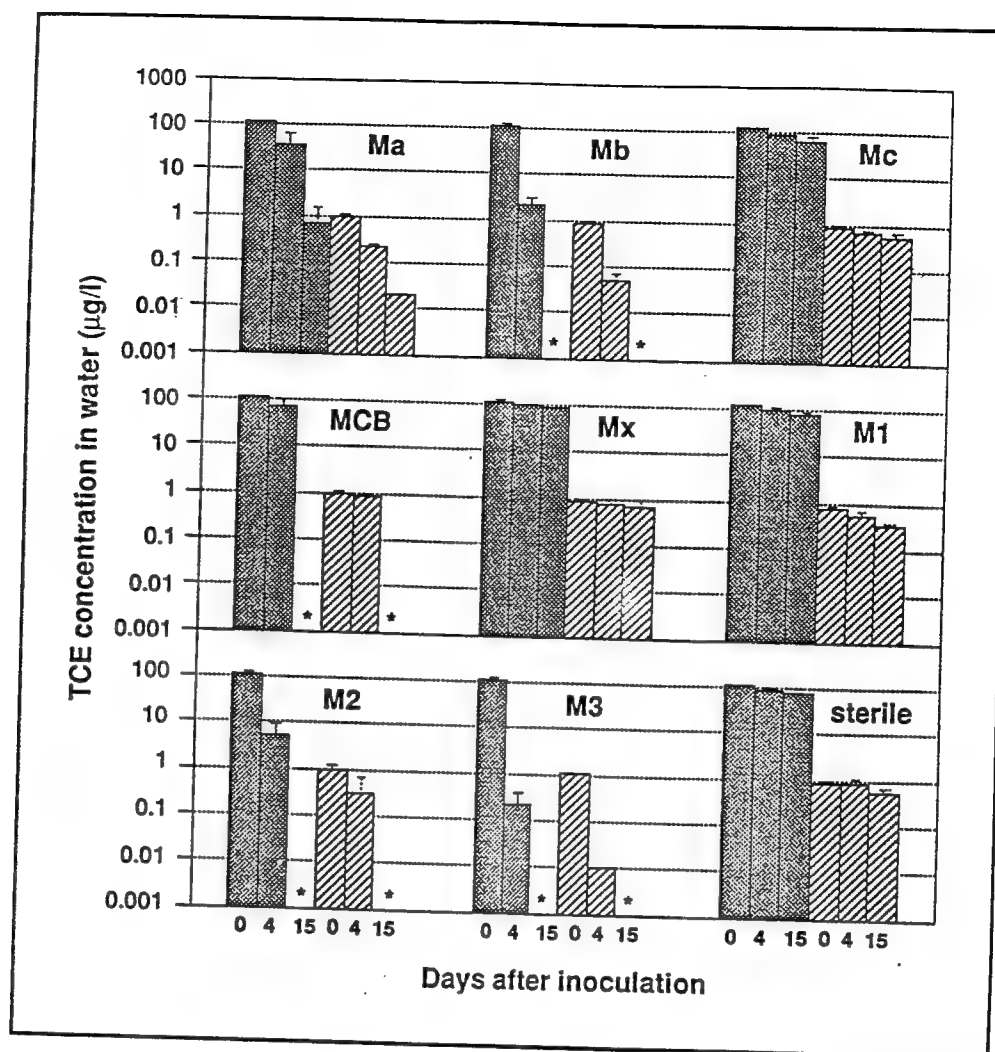
Figure 9. Degradation of 100 and 1.0 µg of TCE per liter by methanotrophs Ma, Mb, Mc, MCB, Mx, M1, M2, and M3 growing on methane. The columns indicate the average amount of TCE present in the experimental bottles at the time of the inoculation, 4, or 15 days later. Legend: ▓ : the initial concentration of TCE was about 100 µg/L; ▨ : the initial concentration of TCE was about 1.0 µg/L; (*): at least two of the four replicates were below the detection limit (about 0.002 µg/L)

Volatile Organic Products Generated During the Cometabolic Degradation of TCE

Some of the experimental bottles used in the above experiment were analyzed by gas chromatography for any halogenated volatile hydrocarbons that could have been produced during TCE degradation. The analysis of experimental bottles that initially contained 100 µg of TCE per liter of water were compared to the analysis of bottles that contained 1 µg of TCE per liter. The analysis was performed by injecting 500 µL of the bottle headspace in the same GC column that was used for TCE detection.

28

The ECD is very sensitive to halogenated compounds, and it would detect volatile chlorinated compounds that could have been produced from TCE degradation. The analysis was made 17 days after the bottles were inoculated.

Concentrated sulfuric acid (0.8 mL) was added to some of the bottles to be analyzed because some chlorinated carboxylic acids have been reported to be products of TCE degradation by methanotrophs. The low pH was expected to facilitate their partitioning of these carboxylic acids to the headspace of the bottle. The retention time of potential degradation products of TCE in the GC column could have been quite important; therefore, the temperature of the oven was gradually increased after 2.3 min from 55 °C to 240 °C at a rate of 20 °C per minute, and the analysis was performed for about 20 min.

The results obtained by analyzing some bottles inoculated with strain M2 are shown in Figure 10. The chromatograms, which were almost identical for bottles initially containing 100 or 1 µg/L of TCE, suggest that volatile compounds were not present.

Similarly, no significant differences were observed between the chromatograms of bottles initially containing 100 or 1 µg of TCE per liter that had been inoculated with strain M3 and into which sulfuric acid was added (Figure 11). In addition, the chromatograms of Figure 11 were similar to those of Figure 10. The very small peaks with a retention time of 2.090 and 2.099 min on the two chromatograms of Figure 11 are those of TCE, and their sizes are consistent with TCE concentrations of about 0.003 and 0.002 µg/L in water, respectively. TCE peaks are not visible in the chromatograms of Figure 10 because the concentration of TCE remaining in those bottles was below the detection limit.

These results indicate that no volatile chlorinated compounds originating from TCE degradation were detected in the headspace. However, the microorganisms had 15 days to degrade TCE and perhaps some of its degradation products.

Bioreactor Design

A preliminary attempt was made to build a bench-top bioreactor that would purify air contaminated with low concentrations of TCE. The bioreactor is depicted in Figure 12 and consists essentially of a modified chromatography column (C) (510 mm long, i.d. 25 mm, Ace Glass, Vineland, NJ) that contains either a suspension of bacterial cells in water or a packing material coated with the cells. The contaminated air stream is introduced into the bottom of the column by using a long glass tube that is tightly fit at the top of the column. At the bottom of that tube is a porous fritted-glass sparger. The contaminated gas is prepared by passing air through a small column (A) packed with granular activated carbon loaded
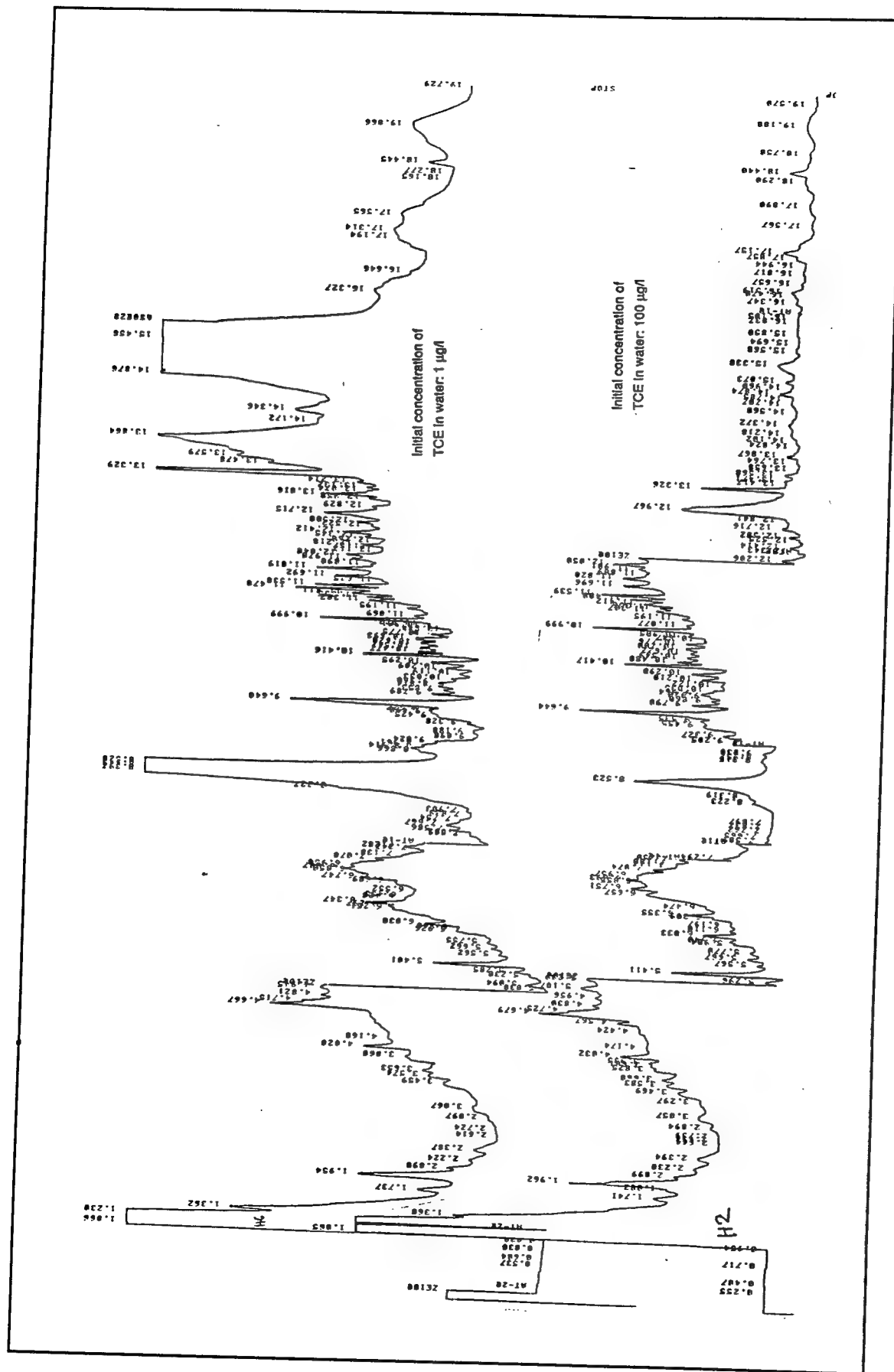
Figure 10.   Chromatograms of the analysis by GC of experimental bottles 17 days after their inoculation with strain M2
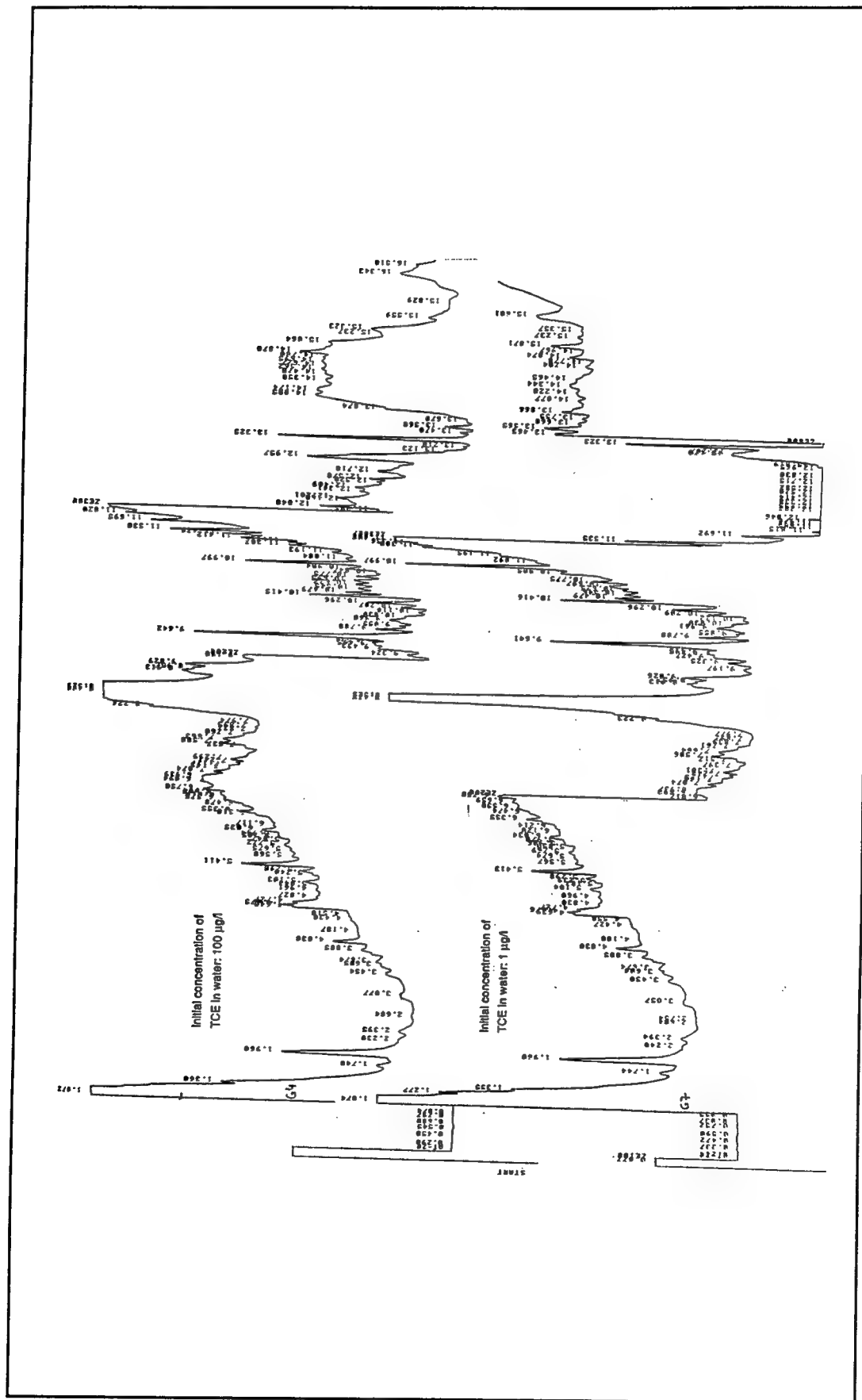
Figure 11. Chromatograms of the analysis by GC of experimental bottles 17 days after their inoculation with strain M3, and after their acidification with sulfuric acid
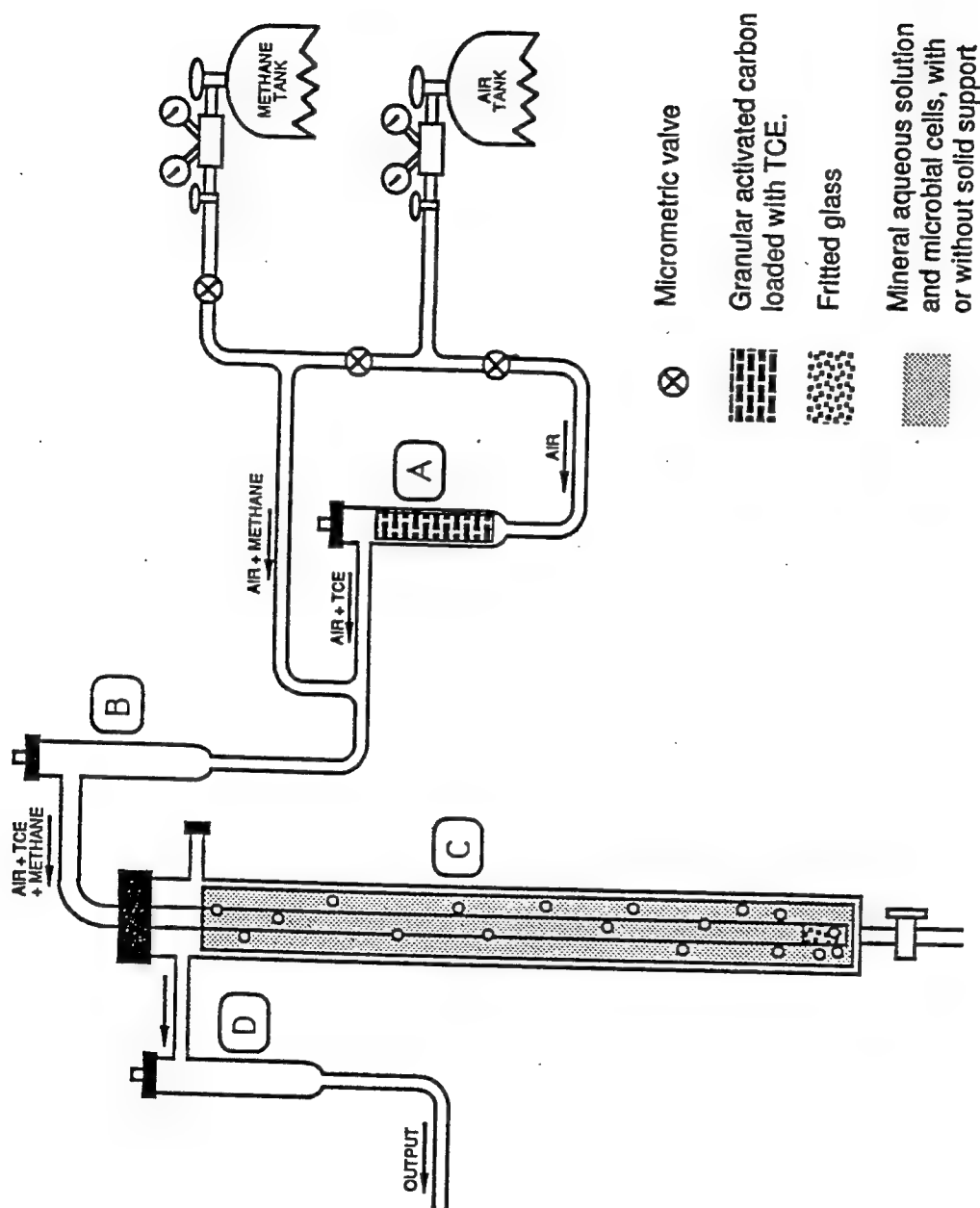
Figure 12.   Initial bioreactor design

with 10 percent (by weight) of TCE. The air emerging from the column contains approximately 4 mg/kg of TCE.

The bioreactor contains additional tubing to permit further dilution of the contaminated air and to permit, when desired, the introduction of methane into the air stream. The flow rates of the gases are controlled by micrometric valves. The analysis of the incoming and outgoing gas is made by using modified test tubes (B and D) closed by Mininert teflon valves through which the gas is sampled with a gas-tight syringe. Analysis is by gas chromatography. All material in contact with the contaminated gas is either glass or teflon.

Because the input and output of the carbon source and the toxic compound occur in the gas phase, their flow does not wash out the microbial cells present in the liquid phase, which would occur in a conventional chemostat in which the carbon source and toxic compound are carried by a stream of liquid. As a consequence, in such a bioreactor, a packing material to which microorganisms are sorbed is not required to prevent the wash-out of microbial cells due to the flux of the carbon source and the toxic compound. Such a packing material may be needed if it is necessary to renew the liquid in the bioreactor at a rate comparable with the growth rate of the bacteria, which might be required to remove water-soluble toxic products that accumulate in the liquid.

The disadvantage of using a liquid suspension of bacterial cells is that the residence time of the bubbles of contaminated gas in the bioreactor is limited approximately to the time necessary for the bubbles to travel from the bottom to the top of the bioreactor. However, such a reactor would not be dried out by high flow rates of contaminated gases. In addition, because the microrganisms are suspended in the aqueous solution, they can easily be replaced with fresh cultures, and the composition of the aqueous phase can be precisely controlled.

If the bioreactor is filled with solid packing that is submerged in the aqueous solution, the residence time of the bubbles of contaminated gas will be somewhat similar to the residence time in the absence of a solid packing. In addition, the previous experiment in which different packing materials were tested did not suggest that the presence of packing material would increase the degradation of TCE in a batch system. Rather, the experiment showed that, without preliminary treatment with EDTA, some packing materials decreased the ability of methanotrophs to degrade TCE. Therefore, a bioreactor filled with solid packing submerged in the aqueous solution might not have an appreciable advantage over the same bioreactor without packing.

In a bioreactor containing packing material that is not submerged with water and to which a biofilm of microorganisms is sorbed, the retention time of contaminated gas can be much higher than in a submerged bioreactor because the retention time will be proportional to the volume of gas found between the solid particles. However, such a bioreactor will be

subject to drying, and will be less flexible than a submerged bioreactor without solid packing because the microbial population will be more difficult to replace or modifiy.

Preliminary tests with the bioreactor filled with microorganisms suspended in an aqueous solution suggested that the minimum flow rate of contaminated gas allowed by the micrometric valves (about 2 mL/min) of the initial bioreactor design was too high for a reactor of the volume described here. There was no mixing system in the reactor, but when the flow of contaminated gas was rapid, the advection caused by the rising bubbles was sufficient to mix the reactor volume through its entire length and to prevent the settling of microbial cells. However, this rapid flow rate of gas exceeded the capacity of the bioreactor to degrade TCE efficiently, and the concentration of TCE in the output gas was about the same as in the input gas.

The bioreactor was not developed further, and attention was turned to mathematical simulation (described in the next chapter) to help in the design and in estimating the flow rate required for proper functioning.

4 Kinetics of TCE Degradation by Methanotrophs

A mathematical model was devised to simulate TCE degradation by methanotrophs in a batch system. Two reasons exist for developing such a model. First, the model may be considered as the mathematical translation of known or hypothesized phenomena, and testing the model is a way of evaluating whether those hypotheses are correct and/or if the phenomena described are sufficient to explain or predict TCE biodegradation by methanotrophs. If the model fails, then finding the reasons for its failure will help to increase the understanding of the process. Second, if the model is adequate, it can be used to predict the biodegradation of TCE in different environments, and it would facilitate the scaling-up of bioreactors and selecting conditions that would optimize the process. The model was intended to help the scaling up of the bioreactor described above.

Consideration to how the model was derived is given here in order to provide a basis for considering the biological meaning of the different mathematical expressions of a model.

Kinetic Model Derivation

Kinetic models of biological systems are often applied without being fully derived from biological principles. This approach has the disadvantage that the biological meaning of mathematical expressions may be difficult to understand. For this reason, the model proposed has been derived based on biological principles. However, this derivation is purely informative. As will be discussed later, the mathematical expressions present in the final form of the model have already been described in the literature. In addition, the same mathematical expression may have different biological meanings, and, consequently, several biological reasons may be responsible for a phenomenon described by a mathematical expression.

The transformation of chemical compounds by bacterial cells is a complex phenomenon that depends on many factors. Therefore, a simple mathematical model to describe such transformations can only be derived if some restrictive assumptions are made, and these restrictions will obviously limit the conditions under which the model is applicable. The model derived below will describe the transformation of TCE in a batch system consisting of a closed bottle containing an aqueous solution of inorganic salts, methane, and TCE. Air, as well as methane, will be present in the bottle headspace, and TCE will partition to this headspace. To simplify the derivation of the model, it is derived for a batch system without headspace, and the small modifications necessary to take into account the partitioning of methane and TCE in the headspace will be introduced later in the development.

In methanotrophs, the enzyme sMMO is responsible for the initial oxidation of TCE, and nonenzymatic reactions are probably responsible for the later transformation of the oxidized TCE (TCE epoxide or other species) to other products. Thus, the primary goal of the mathematical model is to find an expression for $E_{tot}(t)$, the concentration (in milligrams per liter) of enzymatically active sMMO present in the medium at time t.

$E_{tot}(t)$ is assumed to be proportional to the concentration of cells in the medium; therefore, if $X(t)$ is the cell concentration (grams per milliliter) in the medium at time t, then $E_{tot}(t)$ is equal to the product of a proportionality factor E multiplied by the cell concentration $X(t)$ (Equation 1).

$$E_{tot}(t) = EX(t)$$

$$(1)$$

E therefore represents the mass of enzymatically active sMMO per unit mass of microbial cells, and it will be considered constant in most of the conditions simulated by the model. However, depending on the medium and the environmental conditions, the cells may contain different amounts of enzymatically active MMO (i.e., different E). For example, it is expected that E will decrease in the cell under conditions in which the growth-limiting factor is no longer methane but another substance needed for growth. To prevent the rate of methane oxidation exceeding the needs of the cell, it is expected that the cell will reduce the amount of active MMO (i.e., will reduce the value of E) either by deactivating the enzyme, by slowing the transcription of its gene, or by any other mechanism.

The oxidation rate of TCE will be assumed to follow classical enzymatic kinetics that take into account the competitive inhibition between TCE and methane for MMO (Equation 2) (Segel 1976).

$$\text{rate of TCE degradation} = \frac{dT}{dt} = -r_T E_{tot} \frac{T}{K_T\left(1 + \frac{C}{K_c}\right) + T}$$

$$(2)$$

36

where

T = TCE concentration in the aqueous phase, g/mL

t = time, hr

r_T = rate constant for TCE, L/hr

E_{tot} = total enzyme concentration, g/mL

K_T = half-saturation constant for TCE, g/mL

C = methane concentration in the aqueous phase, g/mL

K_C = half-saturation constant for methane, g/mL

In Equation 2, and in the following equations, T, C, X, and E_{tot} are functions of time, but for clarity the characters "(t)" have been omitted from the equations (i.e., $E_{tot}(t)$ will be written E_{tot}).

The expression of E_{tot} given by Equation 1 can be introduced into Equation 2 to give Equation 3.

$$\text{rate of TCE degradation} = \frac{dT}{dt} = -r_T E X \frac{T}{K_T \left(1 + \dfrac{C}{K_C}\right) + T} \tag{3}$$

The rate of methane oxidation can be expressed by a similar equation (Equation 4) in which r_T is the rate constant for methane (L/hr). Equation 3 describes the rate of TCE degradation with methane as a competitive inhibitor, and Equation 4 describes the rate of methane degradation with TCE as a competitive inhibitor.

$$\text{rate of methane degradation} = \frac{dC}{dt} = -r_C E X \frac{C}{K_C \left(1 + \dfrac{T}{K_T}\right) + C} \tag{4}$$

Equations 3 and 4 would be applicable to the kinetics of oxidation of TCE and methane if the enzyme sMMO were free in the aqueous solution. However, in these equations, sMMO is inside the microbial cells, and C and T are defined as the concentration of methane and TCE in the aqueous solution outside the cell. Therefore, it is assumed that methane and TCE concentrations inside and outside the cell are in equilibrium. In other words, it is assumed that the rate at which methane and TCE cross the cell wall and membrane is fast compared to the rate of their oxidation. If this assumption is not true, then the rates of TCE and methane oxidation in Equations 3 and 4 must be replaced by more complex equations that take into account the rates of TCE and methane crossing the cell wall and membrane.

It is likely that the concentrations of methane and TCE inside the cell (the concentrations of both compounds in contact with sMMO) will be different from their concentration outside the cell in the aqueous phase (i.e., C and T). Consequently, the values of K_C and K_T in the equations are likely to be different from the values of similar constants obtained in a system in which the enzyme would be free in solution. This statement means that the values of K_C and K_T estimated for the enzyme sMMO extracted from microbial cells will probably be different from the values of K_C and K_T estimated for living microbial cells.

Molecular oxygen and NADH are also needed by MMO to carry on the oxidation of TCE and methane. It is difficult to introduce mathematical expressions describing the interaction and availability of NADH to sMMO, and it is assumed that the supply of NADH to sMMO is not limiting the rate of oxidation of either methane or TCE. Similarly, it is assumed that oxygen will be present in sufficient quantity and does not limit the oxidation rate of either methane or TCE. Both these assumptions will be checked in preliminary experiments.

The variation with time of the cell concentration $X(t)$ is given by Equation 5.

$$\frac{dX}{dt} = -Y \frac{dC}{dt} - \beta X + \xi \frac{dT}{dt}$$

(5)

where

$Y =$ cell yield: mass of cells created by unit mass of methane metabolized for growth, unitless

$\beta =$ maintenance constant, 1/time

$\xi =$ toxicity constant, or mass of cells killed per unit mass of TCE oxidized, unitless

There are several biological interpretations of the terms of Equation 5, each of them corresponding to different biological assumptions. Only the interpretation most common in the literature is given below.

It is assumed that the concentration of cells will increase in proportion to the amount of methane oxidized ($dX/dt = -Y\, dC/dt$). However, it is also assumed that the cells need a certain amount of energy (maintenance energy) to remain alive, and in the absence of methane or TCE, the concentration of active cells is assumed to decrease following first-order kinetics ($dX/dt = -\beta X$). The oxidation of TCE by sMMO produces reactive products that have been observed to damage the microbial cells. Moreover, the oxidation of TCE (and of CO, one of its oxidation products) consumes NADH that could otherwise be used for cell biosynthesis or to provide energy. Therefore, it is assumed that the oxidation of TCE is associated with a proportional decrease in microbial cell concentration, and in the

38

absence of methane and maintenance energy $dX/dt = \xi \, dT/dt$. Equation 5 is obtained by adding the mathematical expressions of all those assumptions.

Using a different derivation, it is also possible to obtain a somewhat different form for Equation 5; i.e., $dX/dt = -Y \, dC/dt - Y\alpha X + \xi \, dT/dt$, in which the constant β is replaced by the product of two constants, one of them being Y. Depending on the results obtained with the mathematical model, this modified form of Equation 5 may be used instead of Equation 5.

The maintenance requirements will probably not remain constant during the entire growth cycle of the cells and at any methane concentration. When there is abundant methane available, the cells may have a different maintenance energy requirement than when methane is scarce. In this last situation, the cells may change their metabolism to conserve energy and metabolites; they may also begin to store energy or utilize nutrient reserves accumulated when methane was abundant, or they may form resting bodies, in which case their metabolism may change completely. Consequently, the presence of the parameter β in Equation 5 is a simplified way to try to take into account the maintenance metabolism of the cells.

Similar to β, the parameter Y (representing the cell yield) may vary at different methane concentrations, and under different environmental conditions. For example, Y will be low if the cells use much of the methane oxidized to store energy, but Y will be high if the cells use the stored energy to supplement the input of energy provided by the uptake of methane.

A simple kinetic model often used to describe microbial growth on a single substrate is Monod kinetics. The presentation below will show how a Monod kinetics model describing the growth of microbial cells on methane can be derived as a simplified case of the above model.

If TCE is absent from the medium ($T = 0$ and $dT/dt = 0$), Equation 4 reduces to Equation 6. In addition, if the cells have no requirement for maintenance metabolism (i.e., $\beta = 0$), Equation 5 reduces to Equation 7.

$$\frac{dC}{dt} = -r_C E X \frac{C}{K_C + C} \tag{6}$$

$$\frac{dX}{dt} = -Y \frac{dC}{dt} \tag{7}$$

These two equations can be combined to form Equation 8 (by eliminating dC/dt). The expression of Monod kinetics is obtained in Equation 9 by replacing $Yr_C E$ in Equation 8 with a single constant, μmax, and by rearranging the result.

$$\frac{dX}{dt} = Yr_C EX \frac{C}{K_C + C}$$

(8)

$$\frac{1}{X} \frac{dX}{dt} = \mu\text{max} \frac{C}{K_C + C}$$

(9)

These transformations indicate that even though the kinetic model of Equations 3, 4, and 5 was derived under quite limiting assumptions, it is still more general than Monod kinetics.

The kinetic model of Equations 3, 4, and 5 was derived for the particular case when no headspace was present in the system. The modifications necessary to take into account the presence of headspace in the system and the partitioning of methane and TCE to this headspace are given in Equations 10, 11, and 12 below. The only assumption made is that the equilibrium of methane and TCE between the aqueous and gaseous phases is fast compared to the degradation process. This assumption is reasonable if the system is shaken throughout the test period, in which case the equilibrium will be reached within minutes.

$$\frac{dC}{dt} = \frac{-V_l}{V_l + H_C V_a} r_C EX \frac{C}{K_C \left(1 + \frac{T}{K_T}\right) + C}$$

(10)

$$\frac{dT}{dt} = \frac{-V_l}{V_l + H_c V_a} r_T EX \frac{T}{K_T \left(1 + \frac{C}{K_C}\right) + T}$$

(11)

$$\frac{dX}{dt} = -Y \frac{V_l + H_c V_a}{V_l} \frac{dC}{dt} - \beta X + \xi \frac{V_l + H_T V_a}{V_l} \frac{dT}{dt}$$

(12)

In Equations 10, 11, and 12, the definitions of T and C remain the same as in previous equations (concentrations in the aqueous phase), but the new parameters are as follows:

V_l = volume of liquid, mL

H_C = Henry's constant of methane, unitless

V_a = volume of air in contact with the liquid, mL

H_T = Henry's constant of TCE, unitless

40

It is possible to reduce this system of three differential equations to one differential equation (Equation 10) and two equations expressing T and X as functions of C (Equations 13 and 14).

$$\frac{T}{T_o} = \left(\frac{C}{C_o}\right)^{\theta} \qquad \theta = \frac{(V_l + H_c V_A) K_C r_t}{(V_l + H_T V_a) K_T r_C} \tag{13}$$

$$X - X_o = \frac{V_l + H_c V_a}{V_l}\left[\left(-Y + \frac{\beta}{r_C E}\right)(C - C_o) + \frac{\beta K_C}{r_C E}\ln\left(\frac{C}{C_o}\right)\right] \tag{14}$$

$$+ \left(\frac{\beta}{r_T E} + \xi\right)\frac{V_l + H_T V_a}{V_l}(T - T_o)$$

Three additional parameters were introduced in Equations 13 and 14:

T_o = initial TCE concentration, g/mL

C_o = initial methane concentration, g/mL

X_o = initial cell concentration, g/mL

Equations 10, 13, and 14 are then solved using numerical methods. The results are theoretical predictions of the concentrations of methane ($C(t)$), TCE ($T(t)$), and cells ($X(t)$) in the aqueous phase at different times following the addition of methane, TCE, and microbial cells in a batch system. Consequently, the solution of the system of equations can be written as follows:

$$C(t) = C\,(T; r_C, r_T, K_C, K_T, E, Y, \beta, \xi;$$
$$V_a, V_l, H_C, H_T; C_o, X_o, T_o) \tag{15}$$

$$T(t) = T\,(t; r_C, r_T, K_C, K_T, E, Y, \beta, \xi;$$
$$V_a, V_l, H_C, H_T; C_o, X_o, T_o) \tag{16}$$

$$X(t) = X(t; r_C, r_T, K_C, K_T, E, Y, \beta, \xi;$$
$$V_a, V_l, H_C, H_T; C_o, X_o, T_o) \tag{17}$$

These relations indicate that the solutions of the system of Equations 10, 13, and 14 require the knowledge of eight parameters to be estimated (r_C, r_T, K_C, K_T, E, Y, β, and ξ), four parameters that are known (V_a, V_l, H_C, and H_T), and three initial conditions (C_o, X_o, and T_o). Once the eight parameters are known, any type of experiment can be simulated by

varying the values of the initial conditions, by varying V_a and V_l (which describe the volume of air and liquid in the system), and by varying H_c and H_T (which are sensitive to the temperature).

Review of Some Mathematical Models

The following paragraphs give a brief literature review of a few mathematical models describing the cometabolic degradation of a substrate by bacterial cells. To facilitate comparison between the models, the equations for these models are written using the same variables and parameters as the model of Equations 10, 13, and 14. In addition, to simplify the discussion of the models, some details are sometimes omitted in the description. The proposed model described by Equations 10, 13, and 14 is rewritten in a simpler way and called Model 1 (which consists simply of Equations 3, 4, and 5).

$$\frac{dC}{dt} = -r_C E X \frac{C}{K_C \left(1 + \frac{T}{K_T} \right) + C}$$

$$\frac{dX}{dt} = -Y \frac{dC}{dt} - \beta X + \xi \frac{dT}{dt}$$

Model 1

$$\frac{dT}{dt} = -r_T E X \frac{T}{K_T \left(1 + \frac{C}{K_C} \right) + T}$$

The variables and parameters used in the models are defined below. Several interpretations are sometimes given for the parameters, and some of the parameters were already introduced and defined in the derivation of the model, but they are given a more general definition which remains consistent with their initial definition.

$t =$ time

$C =$ methane or concentration of the growth substrate in the aqueous phase

$r_C =$ rate constant for methane

$E =$ mass of enzymatically active sMMO per unit mass of microbial cells, or (for Model 3) fraction of cells that are metabolically active for degrading either the growth substrate (C) or the cometabolic substrate (T)

$K_C =$ half-saturation constant for methane

$T =$ TCE or concentration of the cometabolic substrate in the aqueous phase

$K_T =$ half-saturation constant for TCE

$Y =$ cell yield: mass of cells created by unit mass of methane metabolized for growth

$\beta =$ maintenance or decay constant

$\xi =$ toxicity constant: mass of cells killed per unit mass of cometabolic substrate oxidized, or additional demand exerted by cometabolism on cell metabolism

$r_T =$ rate constant for TCE

$O =$ oxygen concentration in the aqueous phase

$X =$ concentration of microbial cells in the aqueous phase

$r_O =$ rate constant for oxygen

$K_O =$ half-saturation constant for oxygen

$F, G, H, J =$ additional constants or groups of constants needed in some of the models (given without description of their meaning)

Following this notation, dC/dt, dT/dt, and dX/dt describe the variation with time in concentrations in the aqueous phase of methane, TCE, and microbial cells, respectively. Because the purpose of the model is to predict the variation with time in the concentration of TCE (or other cometabolic substrate) in a biological system, only dT/dt would be needed. However, dT/dt is a function of the concentration of microbial cells in the system, which in turn varies depending on the amount of methane degraded. Therefore, the expression of dX/dt and dC/dt is needed in addition to the expression of dT/dt.

Methane and TCE compete for the active site of sMMO, and to avoid this competition and maximize the rate of transformation of TCE, bioreactor systems are developed in which methane is not present during TCE oxidation. As a consequence, several models were devised to simulate TCE degradation by resting cells of methanotrophs (i.e., in the absence of methane). One of these models was developed by Alvarez-Cohen and McCarty (1991a) to simulate the biodegradation of TCE by a mixed culture of resting cells in a batch system (Model 2).

Model 2 assumes that TCE degradation will follow Monod kinetics, and the model takes into account the toxicity to the cell of the products of TCE oxidation, consistent with another study of the same authors (Alvarez-Cohen and McCarty 1991b). Alvarez-Cohen and McCarty (1991a) introduced in their model the constant T_c, the "transformation capacity," to represent the maximum mass of cometabolized compound that can be transformed per unit mass of resting cells. In Model 2, this constant has been replaced by $x = 1/T_c$. This model does not include a term for maintenance energy. In addition, because methane is absent, dT/dt does not need

to take into account the competitive inhibition of methane, and the oxidation rate of TCE is simply described by Michaelis-Menten kinetics. A reasonable fit was obtained between the experimental data and model predictions, confirming the validity of the concept of "transformation capacity."

$$\frac{dC}{dt} = 0 \ (no \ methane \ present \ within \ the \ system)$$

$$\frac{dX}{dt} = \xi \frac{dT}{dt}$$

Model 2

$$\frac{dT}{dt} = -r_T \frac{XT}{K_T + T}$$

Broholm, Christensen, and Jensen (1992) proposed a model to simulate the degradation of TCE by a mixed culture of methanotrophs in a batch system at 10 °C (Model 3). This temperature is consistent with the low temperatures found in groundwater, and TCE oxidation was slow in their experiments (the experiments were conducted for several weeks). The degradation of methane (dC/dt) was described assuming a competitive inhibition by TCE. Similarly, the degradation of TCE (dT/dt) was described assuming a competitive inhibition by methane. The variation in cell density dX/dt was assumed to be proportional to the amount of methane degraded (dC/dt), and a maintenance constant was also present. However, unlike Model 2 and the proposed model (Model 1), Model 3 did not include an expression for TCE toxicity.

The experiments of Broholm, Christensen, and Jensen (1992) were limited to a study of the correlation between the rate of oxidation of methane and TCE. Their model successfully simulated the data obtained at methane concentrations below 1.8 g/L but failed to simulate the data obtained at 3.2 g of methane per liter, supposedly because the growth conditions changed during the experiments.

$$\frac{dC}{dt} = -r_C \frac{XC}{K_C \left(1 + \frac{T}{K_T}\right) + C}$$

$$\frac{dX}{dt} = -Y \frac{dC}{dt} - \beta X$$

Model 3

$$\frac{dT}{dt} = -r_T \frac{XT}{K_T \left(1 = \frac{C}{K_C}\right)} + T$$

Semprini and McCarty (1991, 1992) devised a model to simulate the degradation of TCE in a semi-confined aquifer in which an indigenous population of methane-utilizing bacteria was stimulated by methane and oxygen addition. Only the latter version of their model (1992) is described here (Model 4).

The model attempts to take into account the consumption of oxygen in the aquifer (dO/dt). Oxygen is provided as a nutrient in the field studies that the model simulates, and its availability is likely to influence the rate of TCE degradation. The expressions of dC/dt or dT/dt in their model seem somewhat empirical and do not appear to be general expressions of enzyme kinetics for multiple substrates (for example see Hammes 1982).

$$\frac{dC}{dt} = -E\, r_C \frac{X C}{K_C \left(1 + \dfrac{T}{K_T} \right)} \frac{O}{K_o + O}$$

$$\frac{dX}{dt} = -Y \frac{dC}{dt} - F \beta X \frac{O}{K_O + O}$$

Model 4

$$\frac{dT}{dt} = -E\, r_T \frac{x T}{K_T \left(1 + \dfrac{C}{K_c} \right) + T} \frac{O}{K_O + O}$$

$$\frac{dO}{dt} = G \frac{dC}{dt} - H \beta X \frac{O}{K_O + O}$$

The authors introduced a parameter (E) to express the fraction of the total population active in the cometabolic transformation. This parameter is assumed to be 1.0 when the population is growing ($dX/dt > 0$) and decreases following a first-order process ($dE/dt = -a\,E$, in which a is a constant) when the population is decreasing ($dX/dt < 0$). It is difficult to determine if the variation of E will really follow such an expression, and this is not supported by any study. However, this parameter has the same mathematical effect as the parameter E in the proposed model (Model 1), in which case it represents the amount of active sMMO present in one unit mass of cells.

The variation in population size dX/dt follows kinetics similar to that of Model 3, except that an additional term [$O/(K_O + O)$] expresses that the cell decay (or maintenance requirements) will be maximum when oxygen is fully available but zero in the absence of oxygen. Alvarez-Cohen and McCarty (1991b) showed that resting methanotrophic cells were inactivated more rapidly when shaken in the presence of oxygen than when unshaken. They proposed that shaking the cells increased their decay through endogenous respiration or predation, both of which probably

require oxygen. Therefore, the term $[O/(K_O + O)]$ would simulate this type of phenomenon. But their experiments were not conducted for more than 24 hr, and it seems reasonable to assume that in a longer time period and in the absence of oxygen, the methanotrophs will slowly die or perhaps form resting structures. Consequently, the expression by Alvarez-Cohen and McCarty (1991b) of dX/dt may be limited to simulations of short time periods. In addition, unlike Model 2, Model 4 does not include a term to take into account the toxicity to the cells of products of TCE degradation.

The variation in oxygen concentration (dO/dt) is assumed to be proportional to the amount of methane degraded, and it is apparently proportional to the amount of oxygen consumed by endogenous metabolism.

Alvarez-Cohen and McCarty (1991b) then introduced this model in a spatial model at one dimension that considered the sorption of TCE onto aquifer solids (linear and reversible sorption) and the diffusion and transport of TCE in the aquifer (convection-dispersion equation). The model simulations agreed well with the field observations.

Chang, Voice, and Criddle (1993) used Model 5 to simulate the biodegradation of an aromatic compound by pure cultures. For example, one strain cometabolized p-xylene while growing on toluene as sole carbon and energy source. This model was described in detail by Criddle (1993). The model fit well with the experimental data, but the introduction of an acclimation period was necessary to provide a better fit of the biomass concentration.

The expression of dC/dt is similar to the one described in Model 3, and it considers the competitive inhibition by TCE of methane uptake.

$$\frac{dC}{dt} = -r_C \frac{XC}{K_C\left(1 + \frac{T}{K_T}\right) + C}$$

$$\frac{dX}{dt} = -Y\frac{dC}{dt} - \beta X + \xi \frac{dT}{dt}$$

Model 5

$$\frac{dT}{dt} = -r_T \frac{XT}{K_T\left(1 + \frac{C}{K_C}\right) + T} + J\frac{dC}{dt}\frac{T}{K_T\left(1 + \frac{C}{K_C}\right) + T}$$

The expression of dX/dt includes a decay constant (β), as in Model 3, and also a toxicity constant (ξ), similar to the model of Alvarez-Cohen and McCarty (1991) (Model 2). The presence of this "toxicity" constant was justified to take into account a loss of biomass due to the consumption of reducing power during the oxidation of the cometabolic substrate

46

(e.g. p-xylene) by the dioxygenase. However, unlike TCE oxidation, mention was not made of the possibility that the products of the oxidation of the cometabolic substrate might be toxic to the cells. The expression of both dC/dt and dX/dt in Model 5, therefore, are similar to the one used in the proposed model (Model 1).

The expression of dT/dt is more complex than in the other models, and it is related to the Luedeking-Piret (LP) model that describes the kinetics of product formation, which combines growth-associated and nongrowth-associated contributions (Bailey and Ollis 1986). The first term of the equation is similar to the expression derived for dC/dt; i.e., a competitive inhibition of the degradation of cometabolic substrate (T) by a growth substrate (C).

The second term of the equation predicts an increase in the rate of degradation of cometabolic substrate at increasing rates degradation of growth substrate dC/dt. Criddle (1993) justifies this term by proposing that higher rates of degradation in the presence of a growth substrate might be attributed to elevated activity of catabolic enzymes when the growth substrate is present (induction) or to higher rates of oxidation of growth substrates compared to the rates for autooxidation of biomass. Nevertheless, it is difficult to understand biologically why the expression of the phenomena just described should have the mathematical formulation given by Criddle (1993). In addition, it is not clear why such an expression is not present in the equation describing the degradation of the growth substrate (dC/dt) because the same dioxygenase was assumed to be responsible for the oxidation of both the growth substrate and cometabolic substrate.

Future Research

Although funding from the Waterways Experiment Station has ended, it is hoped that funds will be found to continue the following investigations. It is proposed to use Model 1 to simulate the biodegradation of TCE by pure cultures of methanotrophs in a batch system at room temperature. One of the purposes is to assess if such a model will simulate TCE degradation in a bioreactor in which most of the environmental parameters can be controlled. Pure cultures of methanotrophs will be used to decrease the probability that factors not included in the model influence the experimental results (e.g., predation by protozoa). The studies described above for Model 2 to 5 do not provide enough information for that purpose, either because the model was too simple (e.g. Model 2), mixed cultures were used (Models 2, 3, and 4), the environmental conditions were not optimal (e.g., low temperature in Model 3, limiting oxygen availability in Model 4), too many external parameters influenced TCE degradation (e.g., movement of nutrients and TCE sorption in Model 4), or TCE degradation was not studied (Model 5).

In addition, even though most of the models described above give good fits to the experimental data, the more complicated models (Models 3, 4, and 5) were not tested under a large range of experimental conditions.

The models assume that their parameters are constant. However, the more parameters included in a model, the higher the probability that the model will fit sets of experimental data obtained under somewhat similar conditions. A thorough investigation of the validity of the assumptions requires an investigation of the ability of the model to fit experimental data obtained under very different and even extreme conditions. However, Models 3, 4, and 5 were not tested under many different conditions, and it is not certain that the parameters obtained to fit the experimental data would remain adequate to fit experimental data obtained under quite different environmental conditions. Model 3 was tested under different starting methane and TCE concentrations, but the experimental conditions were still not adequate to find estimates for all the parameters individually (only the ratio $r_T/K_T T$ could be estimated, not r_T or K_T themselves).

Therefore, the validity of Model 1 to predict TCE degradation by methanotrophs will be tested under very different conditions, and a statistical method will be used to find the parameters that provide the best fit of the model to the experimental data obtained from all the different experiments together. The estimated variance of the parameters obtained from the statistics will provide quantitative information on the validity of the model, and indicate which parameters are most likely to vary in response to different environmental conditions.

The following paragraphs briefly describe the methods that will be used to estimate the eight parameters (r_C, r_T, K_C, K_T, E, Y, β, and ξ) of the kinetic model. Actually, only seven parameters need to be estimated because E cannot be estimated independently of r_C and r_T, and its value will be set to 1.0. The effect of the variation of E on the simulation will be assessed later in the study.

The parameters will be estimated by fitting to the kinetic model the experimental data that will be obtained in batch systems by using weighted least-squares methods. However, there are several problems associated with the determination of the parameters, and some of them are discussed below.

Generally, parameters of a mathematical model describing the variation of an observed variable y at different values of an independent variable x would be obtained by fitting the model to one plot formed by plotting the experimental values of y obtained at different values of x. If N replicates of the experiment are run, the model can be fit to each plot of each replicate separately, and N estimates of the parameters can be obtained. Then an average and a standard deviation of each parameter can be calculated from their N estimates (it is not valid to attempt fitting a model to an "average plot" formed by the average of the N y_i's obtained at similar x_i from each replicate).

However, the model in this proposal has seven parameters to be estimated (actually 10 or more if one considers that the initial values C_o, X_o, and T_o are also parameters of the model that need to be evaluated, and that these three values will be different in different experiments). It is almost certain that no single plot obtained from the replicates of an experiment can give a valid estimate of each parameter, because the least-squares problem will be ill-conditioned (Beck and Arnold 1977). In other words, the estimation of the parameters from one single plot will almost always give enormous uncertainties (variance) in the estimation of some of the parameters, and in most cases, the least-squares problem will have an infinity of solutions.

To circumvent this problem, the parameters will not be estimated by minimizing separately the sum of squares of each replicate of each experiment, as suggested above. Rather, the parameters will be found by minimizing one single sum of weighted-squares (total sum of weighted squares (Total SWS)) formed by adding the sum of squares of all the replicates in all the experiments. In addition, the experiments will be carefully designed so that the minimization of Total SWS will not be ill-conditioned. Weights will be included in the sum of squares to acknowledge the fact that the errors associated with the measurement of methane or TCE are relative and not absolute. The weights will be determined by estimating the inverse variance of the relative error associated with methane and TCE measurements, and plots of residuals will be examined at the end of the calculations to ensure that the weights were correctly chosen.

The minimization of Total SWS will be made using the Levenberg-Marquardt method (Seber and Wild 1989) and will involve repeating numerical calculations to evaluate the solutions of Equations 10, 13, and 14, for each plot of each replicate. A computer program written in Pascal language is being developed to handle the calculations.

The numerical calculations will require the determination of initial estimates of the parameters. These estimations will be made by running experiments under extreme values of methane, TCE, or cell concentrations that permit the use of a simplified version of the mathematical model. The purpose will be to obtain experimental conditions such that some parameters can be obtained by fitting a simplified version of the model to one single plot of an experiment.

It is planned to include all the initial values of the variables C_o, X_o, and T_o in the calculations, and each experiment will have its own initial conditions. These initial values will be treated like additional parameters, and they will increase the complexity of the calculations.

Depending on the time required for the calculations (which will be done with a desktop computer), it may be necessary to simplify the method to estimate the parameters.

Some preliminary experiments will be conducted to estimate the concentration of oxygen in the headspace of the experimental bottles above which oxygen concentration has no influence on the rate of oxidation of methane and TCE. Then, all experiments will be conducted with excess oxygen.

Similar kinds of experiments will be conducted to ensure that in no circumstances the concentrations of inorganic salts will become so low that they influence the oxidation rates of either methane or TCE.

Several experiments then will be conducted to estimate the parameters of the kinetic model. The conditions of each experiment will be carefully chosen in such a way as to avoid that the least-squares calculations be ill-conditioned and that the estimation has the best possible accuracy.

It is expected that some limitations of the model will become apparent during the estimation of its parameters. Those limitations and, more generally, the validity of the assumptions made during the derivation of the model will be studied more closely by conducting additional experiments. If feasible, some modifications of the model will be proposed to circumvent the weaknesses of some assumptions.

The assumptions concerning the nonlimiting availability of NADH to sMMO will be assessed by supplying reducing power to the cells indirectly in the form of formic acid. Methanotrophs cannot grow on formate, but the compound can provide the cell with NADH following the oxidation of formate to carbon dioxide by formate dehydrogenase. An absence of increase in the rate of oxidation of TCE (or methane) following the addition of formate to the medium will be consistent with a nonrate limiting availability of NADH to sMMO.

Several experiments will also be conducted to study how the microbial oxidation of TCE will be affected under specific conditions not simulated by the model, including conditions of limiting oxygen or inorganic nutrient availability. The possible inhibitory effect of ammonia as a competitive inhibitor of MMO (Carlsen et al. 1991) will also be assessed.

All the above experiments will be conducted with the same strain of methanotroph, and additional studies may be done with other pure and mixed cultures of methanotrophs to determine whether the model applies to mixed cultures as well as to other pure cultures. The effect of predation by protozoa will be studied by using eukaryotic inhibitors. Some methanotrophs are not able to oxidize TCE, and the effect of their presence in a mixed culture containing TCE oxidizers will be assessed.

The information from these studies will be used to suggest the conditions under which TCE will be oxidized most efficiently.

Computer Program

The computer program in Pascal language was developed to a point at which it was possible to start its testing by fitting artificial data values. However, much work is still needed to debug it, increase its calculation speed if possible, and make it easier to use. The program as given in Appendix A is therefore not completed, even though it seems to work with simple data sets. The following paragraphs give a general overview of the program. Several of the procedures used in the program were obtained from Press et al. (1989) and were often slightly modified.

The program starts by opening an input file listing primary estimates for the parameters to be estimated by the program. In addition, for each parameter, minimum and maximum values are listed; their purpose is to prevent the calculations from diverging too much during the first iterations. The experimental data are listed in the file after the estimates of parameters.

The parameters giving the minimum Total SWS are estimated iteratively by starting from the primary estimates using the method of Marquardt (Press et al. 1989). Each iteration of the Marquardt algorithm requires calculation of the theoretical values predicted by the mathematical model that correspond to each experimental datum, together with the calculation of the partial derivatives with respect to each of the parameters at each experimental datum. Because the mathematical model is only described by differential equations that cannot be solved analytically, the amount of calculation required for each iteration of the Marquardt algorithm is quite important. The integration of the differential equations of the model were made by using the Burlisch-Stoer method (Press et al. 1989), but the procedure iteratively driving this algorithm had to be modified to prevent the calculations from diverging under certain conditions.

The complete listing of the program, together with an example of an input file, is given in Appendix A. Many comments are added in the listing to help in understanding the program, but the program itself is not complete and still has several bugs. The program was written in Think Pascal 4.0 (Symantec Corporation, Cupertino, CA).

References

Alvarez-Cohen, L., and McCarty, P. L. (1991a). "A cometabolic biotransformation model for halogenated aliphatic compounds exhibiting product toxicity," *Environ. Sci. Technol.* 25, 1381-1387.

_____. (1991b). "Effects of toxicity, aeration, and reductant supply on trichloroetylene transformation by a mixed methanotrophic culture," *Appl. Environ. Microbiol.* 57, 228-235.

Anthony, C. (1982). *The biochemistry of methylotrophs.* Academic Press, London.

_____. (1986). "Bacterial oxidation of methane and methanol," *Adv. Microbiol. Physiol.* 27, 113-210.

Bailey, J. E., and Ollis, D. F. (1986). *Biochemical engineering fundamentals.* 2nd ed., McGraw-Hill, New York.

Beck, J. V., and Arnold, K. J. (1977). *Parameter estimation in engineering and science.* John Wiley & Sons, New York.

Broholm, K., Christensen, T. H., and Jensen, B. K. (1992). "Modelling TCE degradation by a mixed culture of methane-oxidizing bacteria," *Wat. Res.* 9, 1177-1185.

Carlsen, H. N., Joergensen, L., and Degn, H. (1991). "Inhibition by ammonia of methane utilization in methylococcus capsulatus (Bath)," *Appl. Microbiol. Biotechnol.* 35, 124-127.

Chang, M. K., Voice, T. C., and Criddle, C. S. (1993). "Kinetics of competitive inhibition and cometabolism in the biodegradation of benzene, toluene, and p-xylene by two pseudomonas isolates," *Biotechnol. Bioeng.* 41, 1057-1065.

Criddle, C. S. (1993). "The kinetics of cometabolism," *Biotechnol. Bioeng.* 41, 1048-1056.

Crittenden, J. C., Cortright, R. C., Rick, B., Tang, S. R., and Perram, D. (1988). "Using GAC to remove VOCs from air stripper off-gas," *J. Am. Water Works Assoc.* 80, 73-84.

DiSpirito, A. A., Gulledge, J., Shiemke, A. K., Murrell, J. C., Lidstrom, M. E., and Krema, C. L. (1992). "Trichloroethylene oxidation by the membraneassociated methane monooxygenase in Type I, Type II, and Type X methanotrophs," *Biodegradation* 2, 151-164.

Fox, B. G., Borneman, J. G., Wackett, L. P., and Lipscomb, J. D. (1990). "Haloalkene oxidation by the soluble methane monooxygenase from methylosinus trichosporium OB3B: Mechanistic and environmental implications," *Biochemistry* 29, 6419-6427.

Fox, B. G., Froland, W. A., Dege, J. E., and Lipscomb, J. D. (1989). "Methane monooxygenase from methylosinus trichosporium OB3B," *J. Biol. Chem.* 264, 10023-10033.

Gosset, J. M. (1987). "Measurement of Henry's Law constant for C1 and C2 chlorinated hydrocarbons," *Environ. Sci. Technol.* 21, 202-208.

Green, P. N. (1992). "Taxonomy of methylotrophic bacteria." *Methane and methanol utilizers.* J. C. Murrell, H. Dalton, ed., Plenum Press, New York, 23-84.

Hammes, G. G. (1982). *Enzyme catalysis and regulation.* Academic Press, Orlando, Florida.

Henry, S. M., and Grbic-Galic, D. (1991a). "Influence of endogenous and exogenous electron donors and trichloroetylene oxidation toxicity on trichloroethylene oxidation by methanotrophic cultures from a groundwater aquifer," *Appl. Environ. Microbiol.* 57, 236-244.

_____. (1991b). "Inhibition of trichloroethylene oxidation by the transformation intermediate carbon monoxide," *Appl. Environ. Micbrobiol.* 57, 1770-1776.

Henrysson, T., and McCarty, P. L. (1993). "Influence of the endogenous storage lipid poly—hydroxybutyrate on the reducing power availability during cometabolism of trichloroethylene and naphthalene by resting methanotrophic mixed cultures," *Appl. Environ. Micbrobiol.* 59, 1602-1606.

Jeffery, G. H., Bassett, J., Mendham, J., and Denney, R. C. (1989). *Vogel's textbook of quantitative chemical analysis.* Longman Scientific and Technical, London.

Koh, S. -C, Bowman, J. P., and Sayler, G. S. (1993). "Soluble methane monooxygenase production and trichloroethylene degradation by a Type I methanotroph, methylomonas methanica 68-1," *Appl. Environ. Micbrobiol.* 59, 960-967.

Nakajima, T., Uchiyama, H., Yagi, D., and Nakahara, T. (1992). "Novel metabolite of trichloroethylene in a methanotrophic bacterium methylocystis sp. M. and hypothetical degradation pathway," *Biosci. Biotechnol. Biochem.* 56, 486-489.

Newman, L. M., and Wackett, L. P. (1991). "Fate of 2,2,2-Trichloroacetaldehyde (chloral hydrate) produced during trichloroethylene oxidation by methanotrophs," *Appl. Environ. Microbiol.* 57, 2399-2402.

Oldenhuis, R., Oedzes, J. Y., van der Waarde, J. J., and Janssen, D. B. (1991). "Kinetics of chlorinated hydrocarbon degradation by methylosinus trichosporium OB3b and toxicity of trichloroethylene," *Appl. Environ. Microbiol.* 57, 7-14.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1989). *"Numerical recipes in Pascal. The art of scientific computing."* Cambridge University Press, New York.

Seber, G. A. F., and Wild, C. J. (1989). *Nonlinear regression.* John Wiley & Sons, New York.

Segel, I. H. (1976). *Biochemical calculations.* 2nd ed., John Wiley & Sons, New York.

Semprini, L., and McCarty, P. L. (1991). "Comparison between model simulations and field results for in-situ biorestoration of chlorinated aliphatics: Part 1. Biostimulation of methanotrophic bacteria," *Ground Water* 29, 365-374.

_____. (1992). "Comparison between model simulations and field results for in-situ biorestoration of chlorinated aliphatics: Part 2. Cometabolic transformations," *Ground Water* 30, 37-44.

Stanier, R. Y., Ingraham, J. L., Wheelis, M. L., and Painter, P. R. (1987). *General microbiology.* 5th ed, MacMillan, London.

Uchiyama, H., Nakajima, T., Yagi, O., and Nakahara, T. (1992). "Role of heterotrophic bacteria in complete mineralization of trichloroethylene by methylocystis sp. Strain M," *Appl. Environ. Micbrobiol.* 58, 3067-3071.

Walsh, C. (1979). *Enzymatic reaction mechanisms.* W. H. Freeman, New York.

54

Whittenbury, R., and Krieg, N. R. (1984). "Methylococcaceae fam. nov." *Bergey's manual of determinative bacteriology.* Williams and Wilkins, Baltimore, Vol 1, 256-262.

Whittenbury, R., Phillips, K. C., and Wilkinson, J. F. (1970). "Enrichment, isolation and some properties of methane-utilizing bacteria," *J. Gen. Microbiol.* 61, 205-218.

Appendix A
Computer Program

Contents

49.033345	2.6	-1	-1	-1	-1	-1
52.8784187	2	-1	-1	-1	-1	-1
55.7942666	1.4	-1	-1	-1	-1	-1
57.457014	1	-1	-1	-1	-1	-1
60.7307919	0.2	-1	-1	-1	-1	-1

SET2   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Sdata	Meto vers.	TCEo vers.	Cello vers.	Va vers.	VI vers.	
5	1	1	2	2	2	

TimeSample	Met_obs	TCE_obs	Cell_obs	MetSig	TCESig	CellSig
0	4	-1	100	-1	-1	-1
0.45327017	3.6	-1	-1	-1	-1	-1
0.89038831	3.2	-1	-1	-1	-1	-1
1.52003874	2.6	-1	-1	-1	-1	-1
2.12478607	2	-1	-1	-1	-1	-1

SET3   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Sdata	Meto vers.	TCEo vers.	Cello vers.	Va vers.	VI vers.	
8	2	1	2	1	1	

TimeSample	Met_obs	TCE_obs	Cell_obs	MetSig	TCESig	CellSig
0	0.02	-1	100	-1	-1	-1
15.9046115	0.018	-1	-1	-1	-1	-1
31.3827736	0.016	-1	-1	-1	-1	-1
54.7909271	0.013	-1	-1	-1	-1	-1
79.9585203	0.01	-1	-1	-1	-1	-1
109.610773	0.007	-1	-1	-1	-1	-1
134.757759	0.005	-1	-1	-1	-1	-1
239.95195	0.001	-1	-1	-1	-1	-1

SET4   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Sdata	Meto vers.	TCEo vers.	Cello vers.	Va vers.	VI vers.	
8	3	1	3	2	2	

TimeSample	Met_obs	TCE_obs	Cell_obs	MetSig	TCESig	CellSig
0	0.2	-1	10	-1	-1	-1
0.44685132	0.18	-1	-1	-1	-1	-1
0.91063316	0.16	-1	-1	-1	-1	-1
1.65238176	0.13	-1	-1	-1	-1	-1
2.48172788	0.1	-1	-1	-1	-1	-1
3.47070542	0.07	-1	-1	-1	-1	-1
4.30248463	0.05	-1	-1	-1	-1	-1
7.64232658	0.01	-1	-1	-1	-1	-1

```
program TCE;
  const

      numb_obs = 20; {the number of "time" observation expected, in ONE set}
{including time zero. BUT IN A SINGLE SET ONLY. i.e. it }
{is the max # of elements of TimeSample}
nvar = 1;  {the maximum number of differential equations solved simultaneously}
{i.e. the max. size of the vectors y and dydx.}
      nstepp = 200;
        {maximum number of intermediate steps in x stored. (612)}
      numb_const = 60;        {the max number of constants used by the model}
{NOTE that I only need room for one version of Meto, Cello, TCEo. }
{Because in the XMatDeriv the different Meto's are in the same }
{column (idem for TCEo and Cello.}
{%%% for rzextr %%%}
      Rzextrlmax = 11; {must be equal to imax inside of bsstep}
      RzextrNmax = 10;   {must be equal to nvar}
      RzextrNcol = 7;       {must be equal to nuse inside of bsstep}
{%%% for marqmin and related %%%%}
      ndatap = 200;  {the maximum number of observations}
   {IN ALL THE SETS, with the addition of the TCE'S, MET'S, CELL'S, unlike numb_obs}
map = 50;           {the max. numb. of constants to be fitted. I must take}
{into account that I may have different versions of Meto, Cello, TCEo, }
{for each set.}
{%%% for CurveFit}
      itermax = 60; {the maximum # of iterations in the least squares}
{%%% To work with several sets of observations %%%%%}
      maxSet = 20;   {the maximum # of sets expected. It is also the }
              {max # of different initial values Meto, TCEo, and Cello.}
  type
    myreal = extended;
    myinteger = integer;
    InVector = array[1..numb_obs] of myreal;
    {typically the vector containing the observations or their theoretical values}
MatDeriv = array[1..numb_obs, 1..numb_const] of myreal;
{typically the matrix containing the values of the partial derivatives}
 {with respect to each constant corresponding to each observation.}
XMatDeriv = array[1..ndatap, 1..numb_const] of myreal;
{The equivalent of MatDeriv but for all the observations, }
{Met, TCE, and Cell together.}
    RealArrayNVAR = array[1..nvar] of myreal;
    FOROdeintXp = array[1..nstepp] of myreal;
    FOROdeintYp = array[1..nvar, 1..nstepp] of myreal;
{%%% for marqmin and related %%%%}
RealArrayNDATA = array[1..ndatap] of myreal;   {vector of observations to be}
{fitted. NOTE that it is not InVector if I use Mrqmin simultaneously for Met, TCE}
{and Cell. In that case ndatap might be 3 times numb_obs.}
    IntegerArrayNDATA = array[1..ndatap] of integer;
```

A4

```
        RealArrayMA = array[1..map] of myreal;
      StringArrayMA = array[1..map] of string[32];          {the vector of the constants name}
      {If this vector is of size 8 or 12, I get an error when doing "stringof(Meto,i)"}
      {in Define. It might be that even if i is <10, the system want it to be }
          {able to use the maximum value of i.}
    IntegerArrayMFIT = array[1..map] of integer; RealArrayMAbyMA = array[1..map, 1..map] of myreal;
      {the matrix of covariance}
    RealArrayMAby1 = array[1..map, 1..1] of myreal;      {a type needed for GAUSSJ}
     RealArrayNPbyNP = RealArrayMAbyMA;        {those 3 declarations are for }
        RealArrayNPbyMP = RealArrayMAby1;          {compatibility with the procedure}
        IntegerArrayNP = IntegerArrayMFIT;          {GaussJ}
        SetInVector = array[1..maxSet] of ^InVector;
    {SetMatDeriv = array[1..maxSet] of MatDeriv;}
        SetInteger6 = array[1..maxSet,  1..6] of integer;
        SetInteger5 = array[1..maxSet,  1..5] of integer;
        SetInteger3 = array[1..maxSet,  1..3] of integer;
        string32 = string[32];  {the type of a file name}
      var
        k: integer;
        word: string32;
        sort: text;         {the internal name of the output file}
        OutputFile: string32;  {used when I want to output my results to a file}
        screen, simple: boolean;
          { TRUE= I want the output on the screen, and the model used is the simple one}
        time, Macc: myreal;  {used to test MetFromTime}
        ma, mfit: integer;  {ma = the number of constants}
    {mfit = the number of constants to enter the statistic}
    {ExpConst: RecConst;}
        fini: boolean;
        MW, DP: integer;  {The minimum width for writing the output}
          {and the decimal places for writing the output}
      Met, dMdt: RealArrayNVAR;
          {are the y and dydx needed by odeint, }
          {odeint will calculate such a vector for each experimental time}
    {Those variables are all declared as dynamic variables}
        Met_cal, TCE_cal, Cell_cal: ^InVector;
        TimeSample: SetInVector;        {an array of pointers to InVector's}
        Met_obs, TCE_obs, Cell_obs: ^InVector;
            {those are the vectors of observations or theoretical values}
        MetSig, TCESig, CellSig: ^InVector;
            {the vectors of standard deviations corresponding to Met, TCE, and Cell}
        PDMet, PDTCE, PDCell: ^MatDeriv;
    {Matrices of partial derivatives created as dynamic variable because I do not have}
    { enough room, probably in the stack (error page 524, "global data exceeds 32 K....)}
    TimeX, Xobs, Xcal, Xsig: ^RealArrayNDATA;        {time, observations, predicted, sigma of obs.}
    {MetoX, TCEoX, CelloX: ^RealArrayNDATA;}
        {not needed yet}
        PDX: ^XMatDeriv;
    {the 3 just above are the equivalent of the 3 above them, but for}
    {use by mrqmin, and include ALL the observations together in the}
    {same vectors.}
```

```
       SetX: ^IntegerArrayNDATA;
{SetX[i] is the adress of the data forming the iest element of the}
{different vector "X" for mrqmin input. To actually find the adress, }
{I need SetToX matrix which indicates the regions between 2 "i"}
{that correspond to each Set, and within as Set at which Met, TCE,}
{or Cell. SetX is defined at the same time as SetToX in MakeMrqVector.}
       Km, Kt, rm, rt, Y, tox, main: myreal;
          {constants (parameters) to be estimated}
       timeo, TCEo, Meto, Cello, E, Hm, Ht, Va, Vl, teta: myreal;
{Those constants will remain constant, whereas the above "constant" }
{(parameters) will be estimated statistically from the calculations.}
       dMdti, TCEi, Celli, Meti: myreal;
{dMdt, TCE, and Cell, from discrete value of Meti}
{data: integer;}
{number of experimental observations; replaced by SetConst[k,1]}
       n: integer;      {the size of the vector y, typically 1, and <nvar}
       tbeg, tend: myreal;    {the time at beginning and end of an ODEINT call}
       i, j: integer;{food for loops }
       reponse: string[8];
       SigPrecMet, SigPrecTCE, SigPrecCell: myreal;
       {the relative errors associated with the observations. Are }
       {defined in the INPUT file and read in Define.}
     aName: StringArrayMA;
{aName = The vector of strings containing the name of the constants}
{used in the program at a location corresponding to that of "a"}
     a, amin, amax: RealArrayMA;
          {a= the vector of constants.}
          {amin= the minimum acceptable values for a.}
          {amax= the maximum acceptable values for a.}
     lista: IntegerArrayMFIT;
     MMin, MMinAcc: myreal;
          {A first guess for the minimum value that Methane will reach }
          {(i.e. when X = 0), and the minimum accuracy required to }
          {detect this minimum, in fraction of the minimum.}
     q, t: myreal;              {used for some testing with a simple Funcs}
       {%%% for derivative %%%%}
     FirstAccr: myreal;     {1/fraction * the relative amount by which a constant will be }
{increased in DefAllDeriv before the FIRST call to MRQMIN.}
{i.e. RelDelA[i]:= FirstAccr*a[i]. (default about 0.1)}
     fraction: myreal;            {it multiplies the relative increase (or decrease) obtained}
{from a previous call or MRQMIN, and the result will be used to}
{estimate the partial derivatives in DefAllDeriv. (default 0.01).}
     defaultDelta: myreal;   {a default value assigned to delta if it is zero in MetAllDerivs}
     RelDelADefault: myreal; { a default value defined in DefRunCons}
{%%% for Odeint (612)%%%}
     OdeintKmax, OdeintKount: myinteger;
       {OdeintKmax is the maximum number of intermediate steps we want to store}
       {if it is zero, no values are saved}
     OdeintDxsav: myreal; {the distance in x after which we store intermediate results.}
     OdeintXp: ^FOROdeintXp;
```

```
OdeintYp: ^FOROdeintYp;
    {those two arrays are to store intermediate results, which are stored at}
    {intervals = OdeintDxsav.}
    {NOTE: OdeintDxsav and   OdeintKmax must be defined in the main program}
    {before calling odeint}
nok, nbad: integer;
    {%%% for ConsOdeint %%%}
{the accuracy Odeint_"eps" defined by Δo = eps x yscal[i]}
    Odeint_eps: myreal;
{a guessed first step size (Odeint_h1)}
    Odeint_h1: myreal;
{the minimum allowed stepsize (Odeint_hmin)}
    Odeint_hmin: myreal;
        {%%% for rzextr (621) %%%}
    RzextrX: array[1..Rzextrlmax] of myreal;
{rzextr use these external arrays to store "xest" which is the square of the}
{step used during the iest call of the routine. Similarly, it will use the}
{matrix below to store the estimated y calculated with xest during the iest step}
    RzextrD: array[1..RzextrNmax, 1..RzextrNcol] of myreal;
        {%%% for marqmin and related (577) %%%%}
    MrqminOchisq: myreal;
        {= the value of the sum of square at the end of marqmin}
{i.e. it is the MINIMUM sum of square.}
    Mychisq: myreal;
{a "chisq" that keeps the previous value of chisq and is not modified by}
{ Mrqmin. I use it to determine if the iteration was succesful or not}
    MrqminBeta: RealArrayMA;
        {= -1/2* gradient of the sum of square with respect }
        {to the "constants" to evaluate}
    RelDelA: RealArrayMA;
{RelDelA = The vector containing by how much the constants in "a"}
{had been increased (relatively) by the last call of MRQMIN.}
    ndata: integer;
{The number of data that will be fitted together, it may be larger}
{than "data" if I put the observations of methane and TCE together}
    covar, alpha: ^RealArrayMAbyMA;
    chisq, alamda: myreal;
{%%% To work with several sets of observations %%%%}
numbSet, numbMeto, numbTCEo, numbCello, numbVa, numbVl: integer;
    SetConst: SetInteger6;
{SetConst= a matrix which describes the # of data (1st col), the address in "a"}
{ of Meto (2), TCEo(3), Cello(4), Va(5), Vl(6) used by each set (in rows)}
    SetVer, SetVerToA: SetInteger5;
{SetVer= a matrix which describes the version of Meto (1), TCEo(2), Cello(3)}
        {Va(4), Vl(5) used by each set (in rows)}
{SetVerToA= a matrix to make the transition between SetVer and SetConst}
        {in row: the version #; in column: Meto(1), TCEo(2), Cello(3)}
        { Va(4), and Vl(5).}
{For a given version, the element gives the address in "a" of the }
{ constant corresponding to the column heading.}
        {NOTE that these two matrices have just one column less than SetConst}
```

```
{, because the latter has a column for "data" in addition. For this reason,}
{Meto is in position 1 in the 2 matrices, but in postion 2 in SetConst.}
     SetToX: SetInteger3;
{This matrix keeps track for each set of the position of its Met, TCE, and Cell}
 { observations in the larger vector Xobs}

      oldFreeHeap, oldStack: longint;

{Used by HeapStack procedure}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$S InOutput}
{*************************************************** }
   procedure HeapStack
(where: string32);
{Indicate the amount of free memory and Stack space available. In addition}
 {gives the difference between those results and the same obtained the }
 {previous call of HeapStack. For that reason "oldFreeHeap" and "oldStack"}
{(2 GLOBAL variables) must be initialize the first time we call the procedure.}
 var
      MinWid: integer;
      actualFreeHeap, actualStack: longint;
  begin
    ShowText;
    MinWid := 7;
actualFreeHeap := FreeMem; actualStack := StackSpace;
write(where : 14, chr(9), ' Δheap = ', chr(9), oldFreeHeap - actualFreeHeap : MinWid, chr(9), ' Δstack = ',
chr(9), oldStack - actualStack : MinWid - 2, chr(9)); writeln(' Free Heap = ', chr(9), actualFreeHeap : MinWid,
chr(9), ' Stack = ', actualStack : MinWid);
      oldFreeHeap := actualFreeHeap;
      oldStack := actualStack;
  end;    {HeapStack}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%} .
{$P}
  procedure Define; {Its role is to read the value of the constants and }
{of the experimental results in the file "entre", and to define the corresponding}
{ values in the program.}
    var
      entre: text;
      word, mystring: string;
      d, i, j, k, m, t: integer;
      title, oldfile: string;
      Wrect: rect;    {see the loop "with" below to see how it is defined}
      char1, char2, char3, char4: char;
  begin
    SetRect(Wrect, 10, 50, 600, 450);
    {toutes les dimensions sont donnees depuis le coin haut gauche}
 SetTextRect(Wrect); {set the dimensions of the interactive window}
    writeln('Select the INPUT file (<return>)');
    readln;
    Reset(entre, OldFileName('select an input file'));
    {page 334, open a file in read only}
 readln(entre, title);        {first line}
```

A8

```
      if Pos('entre', title) <> 1 then
        begin
  writeln('wrong file or problem');
        writeln('press return and the program will quit');
        readln;
        halt;
      end;
    readln(entre);                    {skip the 2nd line.}
    readln(entre, title);             {3th line}
    if Pos('CONSTANTS', title) <> 1 then
      begin
  writeln('could not find the line starting with "CONSTANTS"');
        halt;
      end;
    char1 := '1';
    repeat
      read(entre, char1)              {skip any character before a tab in the line}
    until ord(char1) = 9;
    readln(entre, ma);
    readln(entre);                   {skip the line with titles}
    m := ma;                         {"m" will be used to create SetVerA below}
    if ma > numb_const then
      begin
        writeln('ma = ', ma : 3, '  is > numb_const = ', numb_const : 3);
  writeln('i.e. there are too many constants, especially:'); writeln('Meto"s, TCEo"s, Cello"s');
        halt;
      end;
{%%%%%%% Start reading the constants: first the position in the constants vector,}
{aName, the name of the constant, aMax, a, aMin.}
    for i := 1 to ma do        {"ma" est une valeur PROVISOIRE typiq. 10}
      begin
        aName[i] := '';
        read(entre, char1);
        repeat
          aName[i] := StringOf(aName[i], char1);
          read(entre, char1);
    {that way the tab will not be included in aName}
until ord(char1) = 9;        {i.e. a tab}
        readln(entre, k, aMax[i], a[i], aMin[i]);
        if k <> i then
    writeln('error in "Define": constant[', i : 2, '] has position ', k : 2); end;    {.. of loop over all the
constants}
    readln(entre);
{%%%% start reading the relative error of the observations. These values}
{are used if numbers <= 0 are input in the Sig matrices}
    readln(entre, title);
    if Pos('ERRORS', title) <> 1 then
      begin
        writeln('could not find the heading "ERRORS" in the INPUT file');
        halt;
```

```
            end;
        readln(entre);
        readln(entre, SigPrecMet, SigPrecTCE, SigPrecCell);
        readln(entre);

    {%%%%% start reading the different versions of Meto, TCEo, Cello, and Va/VI %%%%%}
    {It is also here that I start defining the matrix SetVerToA}
        readln(entre, title);
        if Pos('SET INFO', title) <> 1 then
            begin
    writeln('could not find the heading "SET INFO" in the INPUT file'); halt;
            end;
        readln(entre);
    readln(entre, numbSet, numbMeto, numbTCEo, numbCello, numbVa, numbVI); readln(entre);
        if numbset > maxSet then
            begin
    writeln('numbset = ', numbset : 3, ' is > maxSet = ', maxSet : 3); halt;
            end;
    {%%%%% for each Set I will keep its corresponding TimeSample in memory %%%%%}
        HeapStack('before TimeSample init. in Define');
        for k := 1 to numbSet do
            new(TimeSample[k]);
        HeapStack('after TimeSample init. in Define');
        readln(entre, title);
        if Pos('version#', title) <> 1 then
            begin
    writeln('could not find the heading "version#" for Meto in the INPUT file'); halt;
            end;
        for i := 1 to numbMeto do
            {"i" is the version number as entered in SetVerToA}
            begin
                ma := ma + 1;
                readln(entre, char1, aMin[ma], a[ma], aMax[ma]);
                aName[ma] := stringof('Meto', char1);
                SetVerToA[i, 1] := ma;
            end;
        readln(entre);
        readln(entre, title);
        if Pos('version#', title) <> 1 then
            begin
    writeln('could not find the heading "version#" for TCEo in the INPUT file'); halt;
            end;
        for i := 1 to numbTCEo do
            begin
                ma := ma + 1;
                readln(entre, char1, aMin[ma], a[ma], aMax[ma]);
                aName[ma] := stringof('TCEo', char1);
                SetVerToA[i, 2] := ma;
            end;
        readln(entre);
        readln(entre, title);
```

```
  if Pos('version#', title) <> 1 then
    begin
writeln('could not find the heading "version#" for Cello in the INPUT file'); halt;
    end;
  for i := 1 to numbCello do
    begin
ma := ma + 1;
readln(entre, char1, aMin[ma], a[ma], aMax[ma]); aName[ma] := stringof('Cello', char1); SetVerToA[i, 3] :=
ma;
    end;
  readln(entre);
  readln(entre, title);
  if Pos('version#', title) <> 1 then
    begin
writeln('could not find the heading "version#" for Va in the INPUT file'); halt;
    end;
  for i := 1 to numbVa do
    begin
      ma := ma + 1;
      readln(entre, char1, aMin[ma], a[ma], aMax[ma]);
      aName[ma] := stringof('Va', char1);
      SetVerToA[i, 4] := ma;
    end;
  readln(entre);
  readln(entre, title);
  if Pos('version#', title) <> 1 then
    begin
writeln('could not find the heading "version# for VI" in the INPUT file'); halt;
    end;
  for i := 1 to numbVI do
    begin
      ma := ma + 1;
      readln(entre, char1, aMin[ma], a[ma], aMax[ma]);
aName[ma] := stringof('VI', char1); SetVerToA[i, 5] := ma;
    end;
{%%%%% Start reading the data specific to each set, %%%%}
{it is here that I start defining "SetVer",the 1st column of "SetConst" (data),}
{TimeX, Xobs, and Xsig. The same intermediate vectors Met_obs, TCE_obs, Cell_obs}
{are used for each set.}
  t := 0;
  for k := 1 to numbSet do
    begin       {the first thing is to check I am reading the right line}
      readln(entre); {skip a line before reading the next set}
      if k > 99 then
        begin
writeln('in Define, "k" = ', k : 4, ' is > 99 i.e. more than 99 sets'); halt;
        end;
      d := k div 10;
      if d > 0 then        {...i larger than 9}
        begin
```

```
    char1 := chr(48 + d);
            char2 := chr(48 + k mod 10);
            mystring := stringof('SET', char1, char2);
        end        {reste de la division par 10}
      else

      begin
    char1 := chr(48 + k);
            mystring := stringof('SET', char1);
        end;                {I expect to see SET3 if k := 3 and SET12 if k := 12}
    readln(entre, title);
    if Pos(mystring, title) <> 1 then
      begin
    writeln('could not find in INPUT the line starting with ', mystring : MW); halt;
        end;
    readln(entre);        {skip the line of title for the following variables:}
        {data (1), Metoversion(1), TCEo (2), Cello (3), Va (4) VI (5)}
    readln(entre, SetConst[k, 1], SetVer[k, 1], SetVer[k, 2], SetVer[k, 3], SetVer[k, 4], SetVer[k, 5]);
    readln(entre);
    readln(entre);        {skip the line of title for the following variables}
    for j := 1 to SetConst[k, 1] do
      begin
    read(entre, TimeSample[k]^[j], Met_obs^[j], TCE_obs^[j], Cell_obs^[j]); readln(entre, MetSig^[j], TCESig^[j],
    CellSig^[j]);
    {If the sigma value is <= 0.0, it indicates that no value has been calculated}
    {Therefore a default value is assigned. SigPrecXXX represents the relative}
    {error associated with the measurement of XXX.}
        if MetSig^[j] <= 0.0 then
          MetSig^[j] := SigPrecMet * Met_obs^[j];
        if TCESig^[j] <= 0.0 then
          TCESig^[j] := SigPrecTCE * TCE_obs^[j];
        if CellSig^[j] <= 0.0 then
          CellSig^[j] := SigPrecCell * Cell_obs^[j];
        if j > 1 then
          if (TimeSample[k]^[j] <= TimeSample[k]^[j - 1]) then
          {The times must be in increasing order otherwise the}
          {calculations procedures will crash}
          begin
    writeln('TimeSample[', k : 3, '] ^ [', j : 3, '] <= TimeSample [ k ] ^ [', j - 1 : 3, '] STOP !'); halt;
            end;        {...of test}
      end;      {of loop "j" reading the observations for set "k"}
    {%%%%% Now I will load the observations in the big vectors TimeX, Xobs, Xsig %%%}
    for i := 1 to SetConst[k, 1] do      {loop over all the "time" of Set "k"}
      begin
        if Met_obs^[i] > 0.0 then        {...This is a valid "Met" observation to include in Xobs}
          begin
            t := t + 1;
            SetX^[t] := i;
    TimeX^[t] := TimeSample[k]^[i]; Xobs^[t] := Met_obs^[i]; Xsig^[t] := MetSig^[i];
          end;    {...of test to see if we have a valid Met_obs at [i]}
      end;        {...of Loop I over "data" of one set}
```

A12

```pascal
        SetToX[k, 1] := t;    {record the position of the last Met_obs of set "k"}
        for i := 1 to SetConst[k, 1] do
           begin
     if TCE_obs^[i] > 0.0 then       {...This is a valid "Met" observation to include in Xobs}
      begin
              t := t + 1;
              SetX^[t] := i;
  TimeX^[t] := TimeSample[k]^[i]; Xobs^[t] := TCE_obs^[i]; Xsig^[t] := TCESig^[i];
           end;    {...of test to see if we have a valid TCE_obs at [k,i]}
      end;      {...of Loop I over "data" of one set}
     SetToX[k, 2] := t;    {record the position of the last TCE_obs of set "k"}

           for i := 1 to SetConst[k, 1] do
              begin
     if Cell_obs^[i] > 0.0 then      {...This is a valid "Met" observation to include in Xobs}
      begin
                    t := t + 1;
                    SetX^[t] := i;
  TimeX^[t] := TimeSample[k]^[i]; Xobs^[t] := Cell_obs^[i]; Xsig^[t] := CellSig^[i];
                 end;    {...of test to see if we have a valid Met_obs at [k,i]}
              end;       {...of Loop I over "data" of one set and vector Cell_obs}
           SetToX[k, 3] := t;    {record the position of the last Cell_obs of set "k"}
        end;      {... %%%%  of loop K reading the data from each set  %%%%}
      ndata := t;            {this last t is also the total number of observations.}
 {%%%% The big vectors TimeX, Xobs, and Xsig are now defined %%%%}
 {%%%%%% Now I must define the matrix SetConst, that indicates for each set the }
 {corresponding position in "a" of each constant Meto, TCEo, Cello, Va, and VI%%%%%}
  {The first column (the number of data) is the same in SetVer and SetVerA}
      for k := 1 to numbSet do
        begin
           for i := 1 to 5 do
              SetConst[k, i + 1] := SetVertoA[SetVer[k, i], i]; {in SetConst, i= 1 is data, 2 is Meto, 3 is TCEo, 4
  is Cello, 5 is Va, 6 is VI}
  {In SetVertoA and SetVer, the position is one integer less.}
   {SetVer[k,i] is the version of "i" that the set "k" uses.}
  end;
 {%%%%  Below we define the vector RelDelA %%%}
      for i := 1 to ma do
        begin
  if a[i] = 0.0 then
  RelDelA[i] := RelDelADefault
           {RelDelADefault is a default value defined in DefRunCons}
           else
  RelDelA[i] := FirstAccr * a[i];
 {This vector represents the "accroissement" of the constants from the calculations}
  {of MRQMIN. But it will be needed for DefAllDeriv before the first call of MRQMIN}
 {because the amount by which the "variable" constants will be varied to estimate}
 { numerically the partial derivative depends on RelDelA. Therefore I initialize this}
 {vector with an arbitrary value of "FirstAccr", defined as a constant in the main}
 {program.}
        end;
```

```
      close(entre);     {close the input file}
    end;       {....Define}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}

    procedure DefKmfromA
(k: integer;
                    var a: RealArrayMA);
{Its role is to redefine the constants Km, Kt, rm,.... from the values of the vector a}
{after these value have been modified by MRQMIN. This is necessary because }
{MRQMIN does work with "a" and not with the individual names of the constants, }
 {but the user defined functions only work with the individual names, and need}
 {their new values.}
{"k" is the number of the Set concerned by those constants. This is important}
 {because Meto, TCEo, Cello, Va, and VI are different from each set}
    begin
      Km := a[1];
      Kt := a[2];
      rm := a[3];
      rt := a[4];
      main := a[5];
      tox := a[6];
      Y := a[7];
      E := a[8];
      Hm := a[9];
      Ht := a[10];
Meto := a[SetConst[k, 2]]; TCEo := a[SetConst[k, 3]]; Cello := a[SetConst[k, 4]]; Va := a[SetConst[k, 5]];
VI := a[SetConst[k, 6]];
    end;    {...RedefConst}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
    procedure DefLista
(var lista: IntegerArrayMFIT;
                  var mfit: integer); {an interactive procedure to define the vector "lista" which indicates which
constants}
 {need to be evaluated by MRQMIN. It also provide an opportunity to change the values}
{of the constants}
{NOTE that Va and VI are not part of the constant vector "a". The reason}
 {is that I do not plan to introduce them in the compensation. So why Hm}
 {and Ht belongs to "a"? Just for convenience, otherwise they are not}
 {found in any particular vector.}
    var
      Wrect: rect;
      i, k: integer;
      word: string[2];
      iabon: boolean;
    label
      99;
begin
    iabon := true;
    SetRect(Wrect, 10, 50, 600, 450);
    {toutes les dimensions sont donnees depuis le coin haut gauche}
```

A14

```
SetTextRect(Wrect);  {set the dimensions of the interactive window}
     writeln('defining LISTA  and MODIFYING some constants');
writeln(' <enter> to INCLUDE ALL CONSTANTS'); writeln('"M" to MODIFY and SELECT constants ');
readln(word);
if not ((word = 'M') or (word = 'm')) then begin
          for i := 1 to 8 do
             lista[i] := i;
mfit := ma - 2 - numbVa - numbVI; for i := 9 to mfit do
             lista[i] := i + 2; {by default, I put first all the constants in the statistic,}
{except Hm and Ht (9, and 10) for which  NO DERIVATIVES are available}
 {and which can not be included in lista.}
          goto 99;
       end;    {...of no modifications and default definition of Lista}
     writeln('<enter> = put in LISTA, value NOT MODIFIED');
     writeln('"E" = EXCLUDED from LISTA');
writeln('"M" = MODIFY and included IN LISTA'); k := 0;
     while iabon do
       begin
         for i := 1 to ma do
           begin
writeln('constant ', aName[i], ' = ', a[i] : MW); readln(word);
              if not ((word = 'E') or (word = 'e')) then
             {if we accept the constant, then it is loaded into lista}
              begin
if (word = 'm') or (word = 'M') then begin
writeln('enter new value'); readln(a[i]);
                 end;   {of if M}
   if not ((i = 9) or (i = 10) or (i > ma - numbVa - numbVI)) then {A test to prevent constants Hm, Ht, Va, VI to
enter the stat.}
                 begin
                   k := k + 1;
  lista[k] := i; end;    {of Hm, Ht test}
              end;        {of Not Excluded}
          end;         {for i, loop over all the constants}
        mfit := k;
writeln('these are the constants selected and their values'); writeln('press <enter> if OK, "n" for NO and to
redo the loop');
for i := 1 to mfit do
          writeln(aName[lista[i]], ' = ', a[lista[i]]);
          readln(word);
          if not ((word = 'N') or (word = 'n')) then
            iabon := false
          else
            k := 0;
       end;    {iabon}
99:
  end;    {...DefLista}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure DefRunCons;
```

```pascal
{this procedure defines and changes some constants used by Odeint}
{and MetMin in an interactive way. i.e. constant used to run the program}
label
    100;

  var

    fini: boolean;
    reponse: string;
    Wrect: rect;
  begin
    SetRect(Wrect, 10, 50, 600, 450);
    SetTextRect(Wrect);
    OdeintKmax := 100;          {Max number of steps to save}
    OdeintDxsav :=   0.1;        {Save each step of xsav}
    Odeint_eps := 0.000001;     {accuracy}
    Odeint_h1 := 0.01;          {a guessed first step}
    Odeint_hmin :=   0.000001;   {min. allowed stepsize}
    MMinAcc := 1.0e-6;          {the relative error acceptable for MMin}
    Macc := 1.0e-8;             {the relative error acceptable for SimpleMetfromTime}
    n := 1;                 {for Odeint and Bsstep: the number of partial }
                    {differential eauuations  to be solved}
    MW := 14;               {Minimum width to print the data}
    DP := 6;                {digits after the point}
    FirstAccr := 0.01;      {1/fraction * the relative amount by which a constant will be }
{increased in DefAllDeriv before the first call to MRQMIN.}
{i.e. RelDelA[i]:= FirstAccr*a[i].}
    fraction := 0.0001;        {it multiplies the relative increase (or decrease) obtained}
{from a previous call or MRQMIN, and the result will be used to}
{estimate the partial derivatives in DefAllDeriv.}
    defaultDelta := 1.0e-6; {a default value assigned to delta if it is zero in MetAllDerivs}
    RelDelADefault := 0.0001;    {a default value assigned to RelDelA vector}
        {components in Define when the corresponding constant is zero. It serves}
        {the same purpose as defaultDelta: preventing divisions by zero.}
    writeln('MODIFYING some of the MAIN CONSTANTS ("Y") ?');
readln(reponse);
if (reponse <> 'Y') and (reponse <> 'y') then
    goto 100;     {we go to the end of the procedure}
fini := false;
while not fini do
  begin
writeln('the current value of the number of intermediate steps to save'); writeln('OdeintKmax= ',
OdeintKmax : MW, '  change it ?   Y / < ret > '); readln(reponse);
    if (reponse = 'Y') or (reponse = 'y') then
      begin
        write('enter new value >>>');
        readln(OdeintKmax);
      end;
writeln('the distance Δt after which we save an intermediate step'); writeln('OdeintDxsav= ', OdeintDxsav :
MW : DP, '  change it ?   Y / < ret > '); readln(reponse);
    if (reponse = 'Y') or (reponse = 'y') then
      begin
```

A16

```
                    write('enter new value >>>');
        readln(OdeintDxsav);
              end;
        writeln('the accuracy Odeint_"eps" defined by Δo = eps x yscal[i]'); writeln('Odeint_eps= ', Odeint_eps :
        MW : DP, '   change it?  Y/<ret>'); readln(reponse);
            if (reponse = 'Y') or (reponse = 'y') then
              begin
                write('enter new value >>>');
                readln(Odeint_eps);
              end;
            writeln('a guessed first step size');
        writeln('Odeint_h1= ', Odeint_h1 : MW : DP, '  change it ?   Y / < ret > '); readln(reponse);
            if (reponse = 'Y') or (reponse = 'y') then
              begin
                write('enter new value >>>');
                readln(Odeint_h1);
              end;
            writeln('the minimum allowed stepsize');
        writeln('Odeint_hmin= ', Odeint_hmin : MW : DP, '  change it ?   Y / < ret > '); readln(reponse);
            if (reponse = 'Y') or (reponse = 'y') then
              begin
                write('enter new value >>>');
        readln(Odeint_h1);
              end;
        writeln('The relative error acceptable for MMin');
        writeln('MMinAcc= ', MMinAcc : MW, '  change it ?   Y / < ret > '); readln(reponse);
            if (reponse = 'Y') or (reponse = 'y') then
              begin
                write('enter new value >>>');
        readln(MMinAcc);
              end;
        writeln('The relative error acceptable for SimpleMetfromTime'); writeln('Macc= ', Macc : MW, '  change it ?
        Y / < ret > '); readln(reponse);
            if (reponse = 'Y') or (reponse = 'y') then
              begin
                write('enter new value >>>');
        readln(Macc);
                end;
        writeln('FirstAccr, defined by RelDelA[i]:= FirstAccr*a[i] used in Derivative'); writeln('FirstAccr= ', FirstAccr :
        MW, '  change it ?   Y / < ret > '); readln(reponse);
                if (reponse = 'Y') or (reponse = 'y') then
                  begin
                    write('enter new value >>>');
                    readln(FirstAccr);
                  end;
        writeln('fraction, the fraction of Δa used as "delta" in Derivative'); writeln('fraction= ', fraction : MW, '  change
        it ?   Y / < ret > '); readln(reponse);
                if (reponse = 'Y') or (reponse = 'y') then
                  begin
                    write('enter new value >>>');
```

```
        readln(fraction);
              end;
   writeln('defaultDelta, the default value of delta in "Derivative"'); writeln('defaultDelta = ', defaultDelta : MW, '
   change it ?    Y / < ret > '); readln(reponse);

          if (reponse = 'Y') or (reponse = 'y') then
            begin
               write('enter new value >>>');
   readln(defaultDelta);
            end;
   writeln('RelDelADefault, the default value of RelDelA in "Define"'); writeln('RelDelADefault = ',
   RelDelADefault : MW, '   change it ?    Y / < ret > '); readln(reponse);
          if (reponse = 'Y') or (reponse = 'y') then
            begin
               write('enter new value >>>');
               readln(RelDelADefault);
            end;
   writeln('do you want to change the number of digits displayed?'); writeln('and/or START AGAIN?    Y/
   <ret>');
            readln(reponse);
            if (reponse = 'Y') or (reponse = 'y') then
               begin
   write('enter an integer for the number of digits >>>'); readln(DP);
               end
            else
               fini := true;
         end;
100:
   end;    {...DefRunCons}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure  SelectChoice
(var sort: text;
              var screen, simple: boolean); {If we want to print on the screen, "screen" will be true, and
"sort" undefined.}
{Otherwise "screen" will be false and "sort" will be the pointer to an opened}
{file. The problem is that I can not assign a type as variable}
      var
         word: string[4];
         title: string[36];
         Wrect: rect;
         k: integer;
         allTCEo: myreal;
   begin
      screen := true;
simple := false;
writeln('OUTPUT:  on SCREEN (<return>) or on a FILE ("F") ?'); readln(word);
      if (word = 'f') or (word = 'F') then
         screen := false;
      if screen then
         begin
```

```
                    SetRect(Wrect, 10, 50, 600, 450);
                    SetTextRect(Wrect);  {set the dimensions of the interactive window}
               end;    {... of preparing the screen}
          if not screen then
               {I will print on a new file}
             begin
               writeln('TITLE, less than 36 characters');
               readln(title);
               Rewrite(sort, title);
             end; {... of naming and opening the output file}
          for k := 1 to numbTCEo do
             allTCEo := a[SetConst[k, 3]];
          if (main = 0) and (allTCEo = 0.0) then
             begin
               simple := true;
               writeln('main = 0.0 and all the TCEo =   0.0 => the SIMPLE MODEL will be used');
               writeln;
             end
          else
             begin
               writeln('COMPLETE (<return>) or SIMPLE model ("S") ? ');
               readln(word);
               if (word = 's') or (word = 'S') then
                  simple := true;
             end;
     end;      {SelectChoice}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
     procedure CheckConst
(var a: RealArrayMA;
                  var aName: StringArrayMA;
                  ma: integer);
{its role is to insure the constants have appropriate values before }
{doing heavy calculations. This procedure will be called each time}
{the constant vector "a" is changed}
{None of the constants may be smaller than zero, and several can not be}
{equal to zero, otherwise they will crash the program.}
     var
       i, prob: integer;
       problem: IntegerArrayMFIT;
     begin
      prob := 0;
      for i := 1 to ma do
        begin
if a[i] <= 0.0 then
{all the constants susceptible to give problems are those <= 0.0}
          if not ((a[i] = 0.0) and (aMin[i] < 0)) then {...the only cases which are OK are those inside the
statement of not.}
{As a code, aMin[i] < 0 means that a[i] can be = to 0. But no constants}
{at all are allowed to be < 0.i.e. it is a CODE}
```

```
                    begin
      prob := prob + 1; problem[prob] := i;
                  end;

            end;     {... of loop over the constants}

         if prob > 0 then

      {...we have constants values which will crash the program}
            begin
              for i := 1 to prob do
        writeln('constant ', aName[problem[i]] : MW, ' = ', a[problem[i]] : MW); writeln('These values will crash the
      program');
            halt;
          end;
      end;        {... CheckConst}
      {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
      {$P}
        procedure OUTconstants
      (screen: boolean;
                    var sort: text;
                    var a: RealArrayMA;
      var aName: StringArrayMA; ma, MW, DP: integer);
      {OUTconstants will print the value of the constants, either on the screen if screen}
      {is TRUE, or in a file named whose pointer is "sort" if screen is FALSE.}
       {a= the vector of constants values,}
      {aName= the vector of constant names,}
      {ma= the total number of constants.}
      {NOTE, the file must be opened before calling the procedure, (see OUTCovar)}
      {ALSO, I do not output the Va's, and VI's}
          var
          Wrect: rect; {_____%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
        procedure PrintConstants (var fichier: text);
          var
            i: integer;
        begin
          writeln(fichier);
          writeln(fichier);
      writeln(fichier, 'constants of the model');
          writeln(fichier);
          for i := 1 to ma do
        writeln(fichier, aName[i] : MW, chr(9), a[i] : MW : DP); end;        {... of the internal procedure
      PrintConstants}
      {_____%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
        begin        {... of OUTconstants}
          if not screen then
            {I will print on the file "sort"}
            PrintConstants(sort)
          else        {I will print on the screen}
            begin
              SetRect(Wrect, 10, 50, 600, 450);
      {toutes les dimensions sont donnees depuis le coin haut gauche}
      SetTextRect(Wrect);    {set the dimensions of the interactive window}
```

```
    PrintConstants(output);
        end;    {... of printing on screen}
    end;        {...OUTconstants}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure OUTcovar
(screen: boolean;
            var sort: text;
var covar: RealArrayMAbyMA; var aName: StringArrayMA; ma, MW, DP: integer);
{This procedure will output the values of the matrice of covariance.}
{INPUT = "covar", the matrix of covariance, "ma" the number of constants (which }
{determines the size of the matrix), "aName" the vector of the constants name.}
{screen= if "true" then the procedure print on the screen, if "false" it prints on a file,}
 {whose name will be "title". Note that internally this file is called "fichier".}
{MW= the minimum width used to print the data.}
{DP= the number of decimal places.}
     var
       Wrect: rect;
{_____%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
     procedure PrintCovar (var fichier: text);
        var
          i, j: integer;
      begin
        writeln(fichier, 'covariance Matrix');
        writeln(fichier);
        for i := 1 to MW do
          write(fichier, ' ');        {write MW spaces}
        write(fichier, chr(9));
        for i := 1 to ma do
      {print the titles}
          write(fichier, aName[i] : MW, chr(9));
        writeln(fichier);
        for i := 1 to ma do
          begin            {print the row [i] of the covariance matrix for Met}
            write(fichier, aName[i] : MW, chr(9));
            for j := 1 to ma do
   write(fichier, Covar[i, j] : MW, chr(9));  {print the element "j" of row "i"}
writeln(fichier);          {go to next row}
          end;            {...of writing row[i]}
     end;  {...internal procedure PrintCovar}
{_____%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
   begin      {... of OUTcovar}
     if not screen then
         {I will print on a file}
       PrintCovar(sort)
     else      {I will print on the screen}
       begin
         SetRect(Wrect, 10, 50, 600, 450);
         SetTextRect(Wrect);    {set the dimensions of the interactive window}
         PrintCovar(output);
```

```pascal
            end;    {... of printing on screen}
         end;    {...OUTcovar}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure OUTPDmatrix
  (screen: boolean;
                  theSet: integer;
                  var sort: text;
                  var TimeSample: InVector;
                  var PDM: MatDeriv;
                  var aName: StringArrayMA;
   var SetConst: SetInteger6; ma, MW, DP: integer);
  {This procedure will output the predicted values of ONE (PDM) of the matrices of}
  {partial derivative: PDMet, PDTCE, or PDCell}
  {Also, this will be done for ONE SET.}
  {If "screen" is TRUE, the results are printed on the screen, if FALSE, they are}
  {printed in a file named as "title" (externally, but called "sort" internally.)}
  {data= the number of observations and the size of those InVector.}
  {NOTE that the procedure does not make any calculations, it just output the content}
  {of the vectors calculated previously.}
  {NOTE, the file must be opened before calling the procedure, (see OUTCovar)}
      var
        Wrect: rect;
  {_____%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
     procedure PrintPredicted (var fichier: text);
       var
         i, j: integer;
       begin
         writeln(fichier);
   writeln(fichier, 'Predicted values for the partial derivatives of Set', theSet : 3); writeln(fichier);
   writeln(fichier, 'Data from Set', theSet : 3); writeln;
         write(fichier, 'time' : MW, chr(9));
         for i := 1 to 8 do
     {print the name of the constants on top of their corresponding PD}
     {NOTE I will never calculate a partial derivative with respect to Hm and Ht}
     {PDM has no columns 9 and 10 corresponding to Hm or Ht, instead those}
     {columns represent Meto, and TCEo. In addition, aName[11] may be a version of }
     {Meto or TCEo instead of Cello.}
       write(fichier, aName[i] : MW, chr(9));
   writeln(fichier, 'Meto' : MW, chr(9), 'TCEo' : MW, chr(9), 'Cello' : MW);
         writeln(fichier);
         for i := 1 to SetConst[theSet, 1] do
           begin
   write(fichier, TimeSample[i] : MW, chr(9));
             for j := 1 to 11 do
                write(fichier, PDM[i, j] : MW, chr(9));
             writeln(fichier);
           end;
      end;       {... of internal procedure PrintPredicted}
  {_____%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
    begin        {... of OUTPDmatrix}
```

```
          if not screen then
              {I will print on a new file}
          PrintPredicted(sort)
      else        {I will print on the screen}
          begin
              SetRect(Wrect, 10, 50, 600, 450);
SetTextRect(Wrect);   {set the dimensions of the interactive window}
  PrintPredicted(output);
          end;    {... of printing on screen}
      end;        {....OUTPDmatrix}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure OUTOneSetObsCal
(var comment: string;
                  screen: boolean;
                  var sort: text;
                  var theSet: integer;
var TimeSample, Met_cal, TCE_cal, Cell_cal: InVector; var Met_obs, TCE_obs, Cell_obs: InVector;
var MetSig, TCESig, CellSig: InVector; MW, DP: integer);
{This procedure will output the predicted values of the vector methane, TCE, and Cell}
{If "screen" is TRUE, the results are printed on the screen, if FALSE, they are}
{printed in a file named as "title" (externally, but called "sort" internally.}
 {data= the number of observations and the size of those InVector.}
{NOTE that the procedure does not make any calculations, it just output the content}
 {of the vectors calculated previously.}
{comment = any comment we want to print before the data. Is a Var parameter, the }
 {best way is to create it like "comment := stringof('Set', k:3);"}
{NOTE, the file must be opened before calling the procedure, (see OUTCovar)}
      var
        Wrect: rect;
{_____%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
      procedure PrintOneSet (var fichier: text);
        var
wi, i, j: integer;
      begin
          j := theSet;
          writeln(fichier);
          writeln(fichier, comment);
          writeln(fichier);
          wi := 10;              {the MW for below}
write(' ' : wi, chr(9), ' data    ' : wi, chr(9), ' Meto    ' : wi, chr(9), ' TCEo    ' : wi, chr(9)); writeln('Cello    ' : wi,
chr(9), 'Va    ' : wi, chr(9), 'Vl    ' : wi);
write('version' : wi, chr(9), 'n.a.' : wi, chr(9), SetVer[j, 1] : wi, chr(9), SetVer[j, 2] : wi, chr(9)); writeln(SetVer[j,
3] : wi, chr(9), SetVer[j, 4] : wi, chr(9), SetVer[j, 5] : wi);
write('value' : wi, chr(9), SetConst[j, 1] : wi, chr(9), a[SetConst[j, 2]] : wi, chr(9), a[SetConst[j, 3]] : wi,
chr(9)); writeln(a[SetConst[j, 4]] : wi, chr(9), a[SetConst[j, 5]] : wi, chr(9), a[SetConst[j, 6]] : wi);
          writeln;
          writeln(fichier, 'Observed and predicted values for methane, TCE, and Cells');
          writeln(fichier);
          writeln(fichier, 'time' : MW, chr(9), 'Met_obs' : MW, chr(9), 'Met_cal' : MW, chr(9), 'MetSig' : MW);
          for i := 1 to SetConst[theSet, 1] do
```

```
            begin
                write(fichier, TimeSample[i] : MW);
    write(fichier, chr(9), Met_obs[i] : MW); write(fichier, chr(9), Met_cal[i] : MW); writeln(fichier, chr(9), MetSig[i]
    : MW);

            end;
        writeln(fichier);
    writeln(fichier, 'time' : MW, chr(9), 'TCE_obs' : MW, chr(9), 'TCE_cal' : MW, chr(9), 'TCESig' : MW); for i := 1
    to SetConst[theSet, 1] do
            begin
                write(fichier, TimeSample[i] : MW);
    write(fichier, chr(9), TCE_obs[i] : MW); write(fichier, chr(9), TCE_cal[i] : MW); writeln(fichier, chr(9),
    TCESig[i] : MW);
            end;
        writeln(fichier);
    writeln(fichier, 'time' : MW, chr(9), 'Cell_obs' : MW, chr(9), 'Cell_cal' : MW, chr(9), 'CellSig' : MW); for i := 1 to
    SetConst[theSet, 1] do
            begin
                write(fichier, TimeSample[i] : MW);
    write(fichier, chr(9), Cell_obs[i] : MW); write(fichier, chr(9), Cell_cal[i] : MW); writeln(fichier, chr(9), CellSig[i]
    : MW);
            end;
        end;        {... of internal procedure PrintOneSet}
    {_____%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
      begin        {... of OUTOneSetObsCal}
        if not screen then
            {I will print on a new file}
          PrintOneSet(sort)
        else        {I will print on the screen}
          begin
            SetRect(Wrect, 10, 50, 600, 450);
    {toutes les dimensions sont donnees depuis le coin haut gauche}
        SetTextRect(Wrect);    {set the dimensions of the interactive window}
        PrintOneSet(output);
        end;    {... of printing on screen}
      end;        {....OUTOneSetObsCal}
    {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
    {$P}
      procedure OUTAllSetObsCal
    (var TimeX, Xobs, Xcal, Xsig: RealArrayNDATA;
    var TimeSample: SetInVector; var SetX: IntegerArrayNDATA; var SetConst: SetInteger6; var SetToX:
    SetInteger3);
    {Will print the observed and calculated values of all the sets}
        var
          k, t, i, m: integer;
          comment: string;
        begin
          t := 0;
    {initializes the observations matrices to -1, and the variance matrices to -1.}
    {Remember that these matrices exist only for one set at a time, unlike}
    {TimeSample which is a SetInVector which include all the TimeSample}
    {corresponding to each Set.}
```

A24

```
        for k := 1 to numbSet do
          begin
            for i := 1 to numb_obs do
              begin
Met_obs^[i] := -1; TCE_obs^[i] := -1; Cell_obs^[i] := -1; Met_cal^[i] := -1; TCE_cal^[i] := -1; Cell_cal^[i] := -1;
MetSig^[i] := -1;
TCESig^[i] := -1; CellSig^[i] := -1;
              end;
            comment := Stringof('Data from Set ', k : 4);
              {A comment to be printed before each set , see below}
            if k = 1 then
              m := 0
            else
              m := SetToX[k - 1, 3];
              {a test necessary for the first call from i:=1 to SetToX[k,1]}
for i := m + 1 to SetToX[k, 1] do              {loop for Met over all the "Met" of Set "k"}
 begin
              t := t + 1;
              Met_obs^[SetX[t]] := Xobs[t];
Met_cal^[SetX[t]] := Xcal[t]; MetSig^[SetX[t]] := XSig[t];
            end;
for i := SetToX[k, 1] + 1 to SetToX[k, 2] do     {loop for TCE over all the "time" of Set "k"}
 begin
              t := t + 1;
              TCE_obs^[SetX[t]] := Xobs[t];
TCE_cal^[SetX[t]] := Xcal[t]; TCESig^[SetX[t]] := XSig[t];
            end;
for i := SetToX[k, 2] + 1 to SetToX[k, 3] do     {loop for Cell over all the "time" of Set "k"}
 begin
              t := t + 1;
              Cell_obs^[SetX[t]] := Xobs[t];
Cell_cal^[SetX[t]] := Xcal[t]; CellSig^[SetX[t]] := XSig[t];
            end;
    OUTOneSetObsCal(comment, screen, sort, k, TimeSample[k]^, Met_cal^, TCE_cal^, Cell_cal^,
Met_obs^, TCE_obs^, Cell_obs^, MetSig^, TCESig^, CellSig^, MW, DP); {Will print the theoretical and
experimental results on the screen if "true"}
    {or in a file that will be named like "title" if "false".}
    end;    {... of loop over all the K sets}
  end;      {.......OUTAllSetObsCal}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure CheckDefine; {Its role is to check if the procedure Define did}
{a good job assigning the data. It will print the variable on the screen}
    var
      GetScreen: boolean;
      word: string;
    k, i, m, wi: integer;
      Wrect: rect;    {see the loop "with" below to see how it is defined}
  begin
    with Wrect do
```

```
        begin
 top := 50; left := 10; bottom := 450; right := 600
    end;

SetTextRect(Wrect);  {set the dimensions of the interactive window}
Showtext; {page 363, it reveals the text window and make it active.}
 writeln(' CHECKING IF THE CONSTANTS ENTERED ARE CORRECT');
 writeln('aName' : MW, chr(9), 'a' : MW, chr(9), 'aMax' : MW, chr(9), 'aMin' : MW);
     for i := 1 to ma do
    writeln(aName[i] : MW, chr(9), a[i] : MW : DP, chr(9), aMax[i] : MW : DP, chr(9), aMin[i] : MW : DP);
 writeln('type <return> to continue');
     readln;
     GetScreen := screen;
     screen := true;
{OUTAllSetObsCal rely on the value of "screen" to know where to print. }
 {In CheckDefine, I want the results on the screen, so I save the original}
 {value and restore it after OUTAllSetObsCal.}
     writeln('TimeX' : MW, chr(9), 'Xobs' : MW, chr(9), 'Xcal' : MW, chr(9), 'Xsig' : MW);
{%%%%% Start printing the data as loaded in the "X" vectors %%%%%%}
     for k := 1 to numbset do
       begin
         writeln('Set', k : 3);
 if k = 1 then m := 0
         else
           m := SetToX[k - 1, 3];
{a test necessary for the first call from i:=1 to SetToX[k,1]}
         for i := m + 1 to SetToX[k, 1] do          {Met data}
           begin
 if i = m + 1 then           {first line}
               writeln('Set', k : 3, ' Methane');
    writeln(TimeX^[i] : MW, chr(9), Xobs^[i] : MW, chr(9), Xcal^[i] : MW, chr(9), Xsig^[i] : MW); end;
         for i := SetToX[k, 1] + 1 to SetToX[k, 2] do    {TCE data}
           begin
             if i = SetToX[k, 1] + 1 then
 writeln('Set', k : 3, ' TCE');
    writeln(TimeX^[i] : MW, chr(9), Xobs^[i] : MW, chr(9), Xcal^[i] : MW, chr(9), Xsig^[i] : MW); end;
         for i := SetToX[k, 2] + 1 to SetToX[k, 3] do    {Cell data}
           begin
             if i = SetToX[k, 2] + 1 then
 writeln('Set', k : 3, 'Cells');
    writeln(TimeX^[i] : MW, chr(9), Xobs^[i] : MW, chr(9), Xcal^[i] : MW, chr(9), Xsig^[i] : MW); end;
         writeln('press <return> to continue');
         readln;
       end;       {... of loop K over all sets}
     screen := GetScreen;
     readln;
     writeln('type   <return>          to continue');
     writeln('OR   "H"    to stop the program');
     writeln('OR   "N "        to call DefLista to change the constants ');
     readln(word);
     if (word = 'N') or (word = 'n') then
```

```pascal
        DefLista(lista, mfit);
      if (word = 'H') or (word = 'h') then
        halt;
    end;    {...of ChecKDefine}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure Initialize;
  begin
    DefRunCons;
{Define some constants and ask if we want to change any of them}
 {Some of those constants are immediately needed by Define}
    Define;
{This procedure will read the constants "a" and observations from the}
 {file of our choice and assign those values in the program so that}
 {they correspond to their definition. WARNING the input file must }
 {have a stricly defined structure for this work to succeed.}
    DefLista(lista, mfit);
{Defines the vector "lista" which list the constants to introduce in the}
 {statistics, and provide an opportunity to change the constants. Also}
 {defines the number of those constants, "mfit".}
    CheckDefine;
{This procedure will print on the screen the constants and observations}
 {as the program has loaded them. Any mistake will be seen at that point.}
 {A mistake in selecting the constants may be corrected by calling Lista.}
    CheckConst(a, aName, ma);
{Will screen all the constants for values <= 0.0 that may crash the program}
    SelectChoice(sort, screen, simple);
{An interactive procedure to determine where the results must be output.}
  end;         {...Initialize}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$S EquaDiff}
{***************************************************** }
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure SimpleMetfromTime
 (var TimeSample, Met_cal: InVector;
        data: integer);
{Will calculate Methane from TimeSample. The model is t = f(Met), and}
{I use a bisection method to find Met = g(t).}
{ Will return the methane concentration for a given time in conditions}
{under which we can use the simplified equation for methane degradation, i.e.:}
    {1- TCEo = zero (No TCE present)}
    {2- main = zero, the equation does not take maintenance into account.}
{The accuracy of the calculation will be somewhat similar to Odeint, see below.}
    const
      jmax = 100;            {limit the number of bisection iteration}
    label
      100;
    var
      dm: myreal;
        {dm is the difference between M2 and M1 in bisection}
```

```pascal
    i, j: integer;              {food for loop}
    M, M1, time: myreal;
    Foncmid: myreal;

    {a value to store the intermediate values of Cell(Mi)}
{_____%%%%%%%%%%%%%%%%%%%%%%%%%}
  function Fonc (Met: myreal): myreal;
    var
      phi, Cell: myreal;
    begin
      phi := (Vl + Hm * Va) / Vl;
      Cell := Cello + phi * Y * (Meto - Met);
    Fonc := (-Km * ln(Met / Meto) + (Km + Cello / Y / phi + Meto) * ln(Cell / Cello)) / (E * rm * Y * (Cello / Y / phi +
    Meto)) - time; end;          {...Fonc}
{_____%%%%%%%%%%%%%%%%%%%%%%%%%}
  begin
    Macc := Odeint_eps;
{I will take the same level of accuracy as Odeint for Bsstep, but Odeint uses}
{it another way, the allowed error being: Δo = eps * (y[i]+Δy[i]), whereas}
{here I use Δo = eps * y[i] only.}
{I will consider as first two values the extreme of methane, i.e. Meto and zero}
{Then those two values will be used together to find the zero using the}
{bisection method.}
    M1 := Meto;
    M := Meto;      {in case time = 0.0}
    dM := 0;
    for i := 1 to data do      {... loop over all the data}
      begin
        time := TimeSample[i];
        if time = 0.0 then
          goto 100;          {this should only occur once. I MUST include a test before}
          {calling this procedure or MetfromTime to insure that the times are in}
          { increasing order otherwise both procedure will crash.}
        dM := M1;      {M1- 0.0}
{I put "M1" and not the more accurate value "M" to take the larger }
{expectation of M. Otherwise I might be below the next methane}
{ calculated if the times are very close together.}
    for j := 1 to jmax do        {iteration for one point}
          begin      {... loop over many iterations}
            dM := dM * 0.5;
            M := M1 - dM;        {I am going at decreasing methane conc. and increasing}
            {time. But if Met decreases, Fonc(Met,time) will increase. At the beginning,}
            {Fonc(Meto) < 0 [zero - time] therefore if Fonca > 0, that means the new}
            {point M is on the other side of the zero.}
            Foncmid := Fonc(M);
            if Foncmid <= 0.0 then        {... M will be the new M1}
              M1 := M;        {Means that Fonc(M) being >= 0.0, M is still "a" M1}
            if (abs(dM) < Macc * M) or (Foncmid = 0.0) then
              goto 100;
          end;          {normally we only reach this end if j=jmax}
{we will not have more than jmax iterations, the loop will stop and }
{output the value obtained so far.}
```

```
          writeln('SimpleMetFromTime, too many bisections: time = ', time : MW, ' Met_cal[', i : 2, '] = ', M :
MW); 100:        {we are done for this iteration}
          Met_cal[i] := M;
       end;       {....of the loop over all the data}
    end;     {... of SimpleMetfromTime}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure SimpleTCECellfromMet
(var Met, TCE, Cell: InVector;
               data: integer);
{INPUT the amount of methane, OUTPUT: TCE, Cell corresponding to Met}
{Typically, I use Met_cal[k], TCE_cal[k], and Cell_cal[k], and data = SetConst[k,1].}
{NOTE that TCE is supposed to be zero in Simple}
     var
       i: integer;
       teta: myreal;
   begin
     for i := 1 to data do
       begin
         if Met[i] / Meto <= 0.0 then
           begin
writeln('Meti/Meto <= 0.0 in SimpleTCECellfromMet the program stops'); halt;
           end;
         teta := (VI + Hm * Va) * Km * rt / (VI + Ht * Va) / Kt / rm;
         TCE[i] := TCEo * exp(teta * ln(Met[i] / Meto));
         Cell[i] := Cello + Y * (Hm * Va + VI) / VI * (Meto - Met[i]);
       end;     {... of loop over all the data}
    end;        {...SimpleTCECellfromMet}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure SimpleDefMetDeriv
(var a: RealArrayMA;
             ma: integer;
var TimeSample, Met_cal: InVector; var PDMet: MatDeriv;
             data: integer);
{a= the vector of constants                    INPUT}
{ma = is the quantity of constants.            INPUT}
{Met_cal= the values calculated by the model.       INPUT}
{PDMet = the matrix of partial derivatives.          OUTPUT}
{This procedure will input "time" and output the corresponding Met, TCE, Cell,}
{and all partial derivatives, corresponding to time.}
{NOTE, Met_cal MUST HAVE BEEN DEFINED BEFORE CALLING THIS ROUTINE}
     var
       i, j, m: integer;
Denom, teta, phi, Met, TCE, Cell, time: myreal;
{time = the independent variable}
{Met, TCE, Cell= the calculated values at "time"}
{ma= the number of constants in vector "a"}
   begin
     for i := 1 to numb_obs do
```

```pascal
        for j := 1 to numb_const do
           PDMet[i, j] :=  0.0;
      for i := 1 to data do

         begin       {"i" is one row corresponding to TimeSample[i]}
           if TimeSample[i] < 0.0 then
               begin
 writeln('TimeSample[', i : 2, ']', ' is < zero in SimpleDefAllDerivs'); halt;
               end;
           time := TimeSample[i];
           Met := Met_cal[i];
           phi := (Vl + Hm * Va) / Vl;
           Cell := Cello + phi * Y * (Meto - Met);
           Denom := (Km / Met + (Km + Cello / Y / phi + Meto) * phi * Y / Cell);
           {The denominator of all partial derivatives is identical}
           Met_cal[i] := Met;
 {1 is Km}
           PDMet[i, 1] := (-LN(Met / Meto) + LN(Cell / Cello)) / Denom;
 {3 is rm}
           PDMet[i, 3] := -E * Y * (Cello / Y / phi + Meto) * time / Denom;
 {7 is Y}
           PDMet[i, 7] := (-E * rm * time * Meto - Cello / Y / Y / phi * LN(Cell / Cello) + (Km + Cello / Y / phi +
 Meto) * phi * (Meto - Met) / Cell) / Denom;
 {8 is E}
           PDMet[i, 8] := -rm * Y * (Cello / Y / phi + Meto) * time / Denom;
 {9 is Meto}
           PDMet[i, 9] := (-E * rm * time * Y + Km / Meto + LN(Cell / Cello) + (Km + Cello / Y / phi + Meto) * phi *
 Y / Cell) / Denom;
 {11 is Cello}
           PDMet[i, 11] := (-E * rm * time / phi + 1 / Y / phi * LN(Cell / Cello) - (Km + Cello / Y / phi + Meto) * (1 /
 Cello - 1 / Cell)) / Denom;
         end;    {... of loop over all the rows}
      end;    {...SimpleDefMetDeriv}
 {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
 {$P}
   procedure SimplePDCellfromPDMet
  (var PDMet, PDCell: MatDeriv;
 var Met: InVector; ma, data: integer);
 {INPUT: Met, PDMet, ma, data}
 {OUTPUT: PDCell}
 {take the matrix of partial derivatives PDMet and calculate the corresponding}
 {matrix PDCell}
      var
        i, m: integer;
   begin
     for i := 1 to data do
        for m := 1 to ma do
        PDCell[i, m] := -Y * (Vl + Hm * Va) / Vl * PDMet[i, m]; end;       {...SimplePDCellfromPDMet}
 {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
 {$P}
   function MetMin
```

```pascal
                        (Macc: myreal): myreal;
{This procedure will find the value of Met when X = 0. Its OUTPUT is "out"}
{its INPUT is Macc, the desired accuracy in fraction of the output value.}
{ HOWEVER, if MMin is < limit, the procedure output MMin = "limit" .}
{It is useless to calculate a value below that with the risk of using it}
{ and getting underflowed.}
    const
      step = 100;              { the number by which M1 will be multiplied for }
{eventually getting Cell(M1) and Cell(M2) of opposite}
{signs. "step" MUST BE > 1.0}
StartGuess = 1.0e-120;          {the first value by which we start to find MMin}
{IT MUST BE < Meto}
      limit = 1.0e-4900;     {MetMin will not look for a smaller value}
      jmax = 40;               {I limit the number of bisection iteration}
    label
      100;
    var
      dm: myreal;
          {dm is the difference between M2 and M1 in bisection}
      j: integer;             {food for loop}
      fact, M, M1, M2: myreal;
{M1, M2: two values of Met such that Celli(M1)*Celli(M2)<0 (opposite signs)}
      {fact= the factor by which M1 will have to be multiplied so that we }
      {eventually get a Celli of opposite sign i.e. we found M2.}
      Cella, Cellb: myreal;
      {two value to store the intermediate values of Cell(Mi)}
      done: boolean;
    function Cell (Met: myreal): myreal;
      var
        a, b, teta, TCEi: myreal;
      begin
        if Met / Meto = 0.0 then
          begin
            writeln('Met/Meto = 0.0 in functon Cell of MetMin');
            MetMin := limit;
            goto 100;
          end;
        teta := (Vl + Hm * Va) * Km * rt / (Vl + Ht * Va) / Kt / rm;
        TCEi := TCEo * exp(teta * ln(Met / Meto));
        a := (-1 + main / rm / E) * (Met - Meto) + (main * Km / rm / E) * ln(Met / Meto);
b := (main / E + rt * tox / Y) * (Vl + Ht * Va) * Y / Vl / rt * (TCEi - TCEo);
        Cell := Cello + Y * (Hm * Va + Vl) / Vl * a + b;
      end;
    begin
{The first goal is to find two values of Celli one negative the other positive.}
{Then those two values will be used together to find the zero using the}
{bisection method.}
      if Meto = 0.0 then
        begin
writeln('Meto = 0.0 in MetMin, the program is stopped'); halt;
```

```
                        end;
            {just a test to prevent problems if Meto is badly assigned}
                M1 := StartGuess;

              Cella := Cell(M1);

             if Cella = 0.0 then
                  begin
        writeln('Cella = 0.0'); MetMin := M1;
                    goto 100;          {we are done EH! (celui qui meprise l'improbable, etc...)}
                  end;
                if Cella > 0.0 then
                  fact := 1 / step;
                {we will decrease M1 by a factor of 1/step until we find a neg. Cell value}
                if Cella < 0.0 then
                  fact := step;
                {we will increase M1 by a factor of step until we find a neg. Cell value}
              done := false;
                repeat
                  M2 := M1;
                  M1 := M2 * fact;
                  if M1 <= limit then
                    begin
                      writeln('Before bisection MetMin <=  ', limit);
        MetMin := limit;
                      goto 100
                    end;
                  Cellb := Cell(M1);
                  if Cellb = 0.0 then
                    begin
        writeln('Cellb = 0.0'); MetMin := M1;
                  goto 100;            {we are done EH! (celui qui meprise l'improbable, etc...)}
                end;
                until Cella * Cellb < 0.0;
        {%%%% Now we use M1 and M2 for a bisection %%%%}
                if fact > 1.0 then      {i.e. fact = step so M2<M1}
                  begin
                    M := M1;
        M1 := M2; M2 := M;
                  end;
                {now M2 > M1, and I know that Cell(M2) > 0, Cell(M1) < 0}
        dM := M2 - M1;
                for j := 1 to jmax do
                  begin
                    dM := dM * 0.5;
                    M := M1 + dM;
        if M / Meto = 0.0 then         {ne devrait pas arriver}
                      begin
                        writeln('MMin is underflowed, it will be set to "limit"');
                        MetMin := limit;
                        goto 100;
                      end;
```

```
          Cella := Cell(M);
  if Cella <= 0.0 then          {... M will be the new M1}
    M1 := M;                    {Means that Cell(M) being <= 0.0, M is still "a" M1}
   if (abs(dM) < Macc * M) or (Cella = 0.0) then
{Macc is the relative accuracy of MetMin so the variation dm must be smaller}
{than Macc*M}
          begin
  if Cella < 0.0 then MetMin := M + dM
              else
                MetMin := M;
{I want the UPPER limit of MetMin, such that Cell will be positive. If Cell}
{is negative, I risk calculating log of negative numbers later}
          goto 100;
          end;
      end;        {normally we only reach this end if j=jmax}
    writeln('pause in MetMin, too many bisections');
    readln;
100:      {we are done}
  end;   {... of Metmin}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure derivs
(t: myreal;
              var Met, dMdt: RealArrayNVAR); {Return the vector of derivatives dMdt at the point (t, Met).
Both dMdt and t}
{are vectors of size n <= nvar. The program will calculate the values of }
 {each variable yi of the vector y at increments of x, and need to be provided}
 {with the gradient dydx which is a function of both y and x. In our particular}
 {case, there is only one y therefore the vectors y and dydx are of size 1}
 var
      a, b, TCEi, Celli: myreal;
  begin
   if MMin <= 0.0 then     {This should not occur unless MetMin had a problem}
     begin
writeln('problem in derivs, MMin <=0.0  = ', MMin, ' the program will stop'); halt;
     end;
    if Met[1] <= MMin then
     dMdt[1] := 0.0
{MMin is calculated as the amount of methane remaining in solution at time}
{going to infinity. Therefore if the numerical procedure attempt to use }
{methane values below MMin, I return the value of dMdt corresponding to}
{very large Met.}
    else
     begin
       teta := (Vl + Hm * Va) * Km * rt / (Vl + Ht * Va) / Kt / rm;
       TCEi := TCEo * exp(teta * ln(Met[1] / Meto));
a := (-1 + main / rm / E) * (Met[1] - Meto) + (main * Km / rm / E) * ln(Met[1] / Meto); b := (main / E + rt * tox / Y) *
(Vl + Ht * Va) * Y / Vl / rt * (TCEi - TCEo);
       Celli := Cello + Y * (Hm * Va + Vl) / Vl * a + b;
       if Celli <= 0.0 then
         Celli := 0.0;
```

```pascal
    dMdt[1] := -Vl / (Hm * Va + Vl) * rm * E * Celli * Met[1] / (Km * (1 + TCEi / Kt) + Met[1]); end          {...of
else if Met > MMin}
    end;     {...of derivs}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure mmid
  (var y, dydx: RealArrayNVAR;              {vector of y and dydx}
                   n: integer; {size of vector y and dydx}
  xs, htot: myreal; {xs is the starting x, xs+htot the endpoint}
  nstep: integer; {dans bsstep sera <= 96}
                   var yout: RealArrayNVAR);  {y(xs+htot)}
  {616}
      var
        step, i: integer;
        x, swap, h2, h: myreal;        {x= the intermediate x.}
        {swap = an intermediate used to assign new value to yn at each step}
        {h2 = 2.0*h, and h= the small step of the discretization of htot.}
        ym, yn: ^RealArrayNVAR;      {intermediate values, yn being a step "h" ahead of ym}
  begin
    new(ym);
    new(yn);
    h := htot / nstep;       {the discretization of htot in "nstep" steps}
    for i := 1 to n do
      begin
        ym^[i] := y[i];
        yn^[i] := y[i] + h * dydx[i]  {first step}
      end;
    x := xs + h;
    derivs(x, yn^, yout); {yout is used for the temporary storage of derivatives. It }
{is not how yout was defined however i.e. we just use yout instead of }
{defining a new variable just for this step.}
    h2 := 2.0 * h;
    for step := 2 to nstep do         {general step}
      begin
        for i := 1 to n do
          begin
            swap := ym^[i] + h2 * yout[i];
            ym^[i] := yn^[i];
            yn^[i] := swap
{ym takes the old value of yn, and yn increases so that it is always a}
{step "h" ahead of ym}
          end;
        x := x + h;            {prepares x for the next step}
    derivs(x, yn^, yout)    {prepare yn and dydx for the next step}
  end;    {... of the cycle through "nstep" steps}
    for i := 1 to n do
    {we are done and will output y(x+ htot)}
yout[i] := 0.5 * (ym^[i] + yn^[i] + h * yout[i]);
    dispose(yn);
    dispose(ym);
```

```
      end;    {... of mmid}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure rzextr
  (iest: integer;
xest: myreal;  {the step used in bsstep to determine y[iest]}
 var yest, yz, dy: RealArrayNVAR;
              n, nuse: integer); {"n"= nseq of bsstep. "nuse" = nuse of bsstep.}
{it is the iest time that we call rzextr from bsstep, and this corresponds to the}
{use of nseq[iest] in bsstep.}
{"yest" = the y[iest] estimate of y by bsstep. "yz" = is the value of y extrapolated}
{by rzextr, i.e. it is an OUTPUT. "dy" = error estimated by rzextr from the difference}
 {between the actual "yz" and the preceding one. This error will be used by bsstep}
 {to estimate wether we have reached the required precision, and wether we can stop }
{This procedure input the estimated yest obtained by bsstep using xest. Then it }
{fits a rational function to all the yest(xest) obtained so far (the xest are stored in }
 {RzextrX vector and some constants of the procedure corresponding to yest are stored}
 {in RzextrD. Both vector and matrix located outside of the routine), and uses this }
 {function to estimate what would be yest for xest tending to zero. In addition, the}
 {routine estimate the error of yest.}
      var
        m1, k, j: integer;         {food for loops}
        yy, v, ddy, c, b1, b: myreal;
        fx: array[1..RzextrNcol] of myreal;
    begin
      RzextrX[iest] := xest; {save the current xest}
      if iest = 1 then            {...this is the first point (yest, xest) to enter rzextr}
        for j := 1 to n do          {for the length of the vector y}
          begin
            yz[j] := yest[j];   {output = input, rzextr can not extrapolate from one point}
            RzextrD[j, 1] := yest[j];   {save the current yest}
            dy[j] := yest[j];        {the error is equal to the value (no way to estimate the error}
          end
      else
        begin
          if iest < nuse then      {... we will take all the points in storage for the curve fit}
            m1 := iest
          else                  {... we will only take the last "nuse" points}
            m1 := nuse;
          for k := 1 to m1 - 1 do
            fx[k + 1] := RzextrX[iest - k] / xest;
          for j := 1 to n do       {evaluate next diagonal in tableau}
            begin
              yy := yest[j];
              v := RzextrD[j, 1];
              c := yy;
              RzextrD[j, 1] := yy;
              for k := 2 to m1 do
                begin
                  b1 := fx[k] * v;
                  b := b1 - c;
```

```
                if b <> 0.0 then
                  begin
```

$$b := (c - v) / b; \; ddy := c \cdot b;$$

```
                    c := b1 * b
                  end
                else      {care needed to avoid division by 0.}
                  ddy := v;
                if k <> m1 then
  v := RzextrD[j, k]; RzextrD[j, k] := ddy;
  yy := yy + ddy end;
                dy[j] := ddy;
                yz[j] := yy
              end
        end
    end;    {... of rzextr}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure bsstep
(var y, dydx: RealArrayNVAR;
                  n: integer;
var x: myreal; htry, eps: myreal;
var yscal: RealArrayNVAR; var hdid, hnext: myreal);
{all those variable have been explained in ODEINT}
{This procedure makes a numerical integration from the vector y(x) to the vector}
{ y(x+htry). The process consists in dividing the big step "htry" in "nseq[i]" equal substep.}
{A value y(x+htry, nseq[i]) is evaluated using this discretization and the prodedure}
{MMID (modified midpoint method 616). Then a new value "nseq[i+1]" is chosen and a}
{new value y(x+htry, nseq[i+1] is calculated. The difference between yi+1 and yi >>>>>>>}
{y= at INPUT this is the starting value y(x), at OUTPUT it is the end value y(x+htry)}
{by bsstep. The program permits to have y as a vector of dimension n if we plan}
{to solve several D.Equations at the same time.}
{dydx= the vector (size n but for me n=1) of the partial derivatives at "time" x}
{n= the size of the vectors y and dydx, in my case n=1}
{x= the independent variable, in my case "time". It is the time at the beginning}
      {of the step, and "y" is actually "y(x)" the output value of "y" will be y(x+htry)}
{htry= the Δx we try to reach. If we do not succeed after "imax" because of too }
            {low accuracy, the procedure reduce htry and try again. This gives}
            {hdid and hnext, see below.}
{eps= The accuracy is defined in absolute value by Δo = eps x yscal[i]}
{yscal= the vector against which the accuracy is compared. yscal = y[i]+ Δy[i].}
            {Odeint takes care of defining yscal before calling bsstep}
{hdid= the step Δx actually used by the procedure to succeed in finding y(x+Δx) }
            {within the accuracy standard.}
{hnext= the proposed next step, the next time we call bsstep from Odeint.}
            { It is not necessarily hdid because to optimize}
{the calculations, hnext could be < or > than hdid.}
    label
      99;
    const
      imax = 11;   {max # of nseq, see below}
```

```pascal
    nuse = 7;           {max # of points (y(x+htry, nseq[i]) taken for the extrapolation}
    shrink = 0.95e0;
    grow =  1.2e0;           {the fractions to decrease or increase the step htry}
  var
    j, i: integer;  {food for FOR and WHILE loops}
    xsav, xest, h, errmax: myreal;
    Maxh: myreal;              {the maximum step to avoid having negative methane values}
{&&&&&& my addition &&&&&&&&}
    ysav, dysav, yseq, yerr: ^RealArrayNVAR;
    nseq: array[1..imax] of integer;
    goforit: boolean;          {when wrong we escape the routine without having finished}
begin
  new(ysav);
  new(dysav);
  new(yseq);
  new(yerr);
  nseq[1] := 2;
  nseq[2] := 4;
  nseq[3] := 6;
  nseq[4] := 8;
  nseq[5] := 12;
  nseq[6] := 16;
  nseq[7] := 24;
  nseq[8] := 32;
  nseq[9] := 48;
  nseq[10] := 64;
  nseq[11] := 96;
  h := htry;    {the big step to make by this run of the procedure}
  xsav := x;    {xsav is the initial value of x. This value will of course remain constant}
          {for all nseq.}
  for i := 1 to n do
    begin     {just saving the starting values}
      ysav^[i] := y[i];
      dysav^[i] := dydx[i]
    end;
  goforit := true;
  while goforit = true do
{this loop turns on indefinitely unless we escape from it either by }
 {GOTO or by displaying an error message, and turning goforit to false.}
 {GOTO will be used when the discretization of h was small enough}
{to permit an estimation of y that is within precision limits.}
 begin
{&&&&&&&&&&&&&&&&&&&  addition &&&&&&&&&&&&&&&&&&&&&&&&}
{Its purpose is to prevent ysav (methane) to be below its minimum value}
{when time go to infinity (MMin). If it were below MMin, then Cell would become}
 {negative, and the corresponding derivative would be foolish. In derivs }
{I have stated that if Met < MMin the procedure will return the value }
 {dMdt = 0.0. For that reason the test below will stop the program if }
 {that happens. Note that MMin calculated by MetMin can not be lower than}
{a certain constant "limit" in this procedure, otherwise there are conditions}
 {under which the actual MMin would be below the smallest real (extended) }
```

```pascal
                {that Pascal supports.}
                    if dysav^[1] = 0.0 then
{to avoid division by zero in the following step}
            begin
        writeln('dysav = 0.0 in bsstep, Maxh can not be estimated, the program stops'); halt;
            end;
        Maxh := (MMin - ysav^[1]) / dysav^[1];
        if h > Maxh then
            h := Maxh;
    {&&&&&&&&&&&&&&&&&&&&  End of  addition &&&&&&&&&&&&&&&&&&&&&&&&&&}
        for i := 1 to imax do
                begin
                    mmid(ysav^, dysav^, n, xsav, h, nseq[i], yseq^);
        {given the big step to make (h), and the number of substeps (nseq[i]), mmid}
        {give "yseq" the estimated y(xsav + h)}
                    xest := sqr(h / nseq[i]);
        {RZEXTR will calculate a curve y(h/nseq[i]) for different nseq (it keeps }
        {the "y" obtained at previous nseq in memory) and will extrapolate the value}
        { y(lim nseq->inf) each time we add a larger nseq. In reality, because the}
        {error of mmid is in sqr(n/nseq) the routine use xest as variable. In addition}
        {the routine will output "yerr", an estimation of the error of the estimation.}
        {It is estimated by comparing the estimated y(htry) obtained at two following }
        {steps.}
                    rzextr(i, xest, yseq^, y, yerr^, n, nuse);
                    if i > 3 then      {we enter the loop in which a possibility to stop exist, but to}
        {prevent this to occur too soon due to a suspect early }
        {convergence, this is only possible if i>3.}
                        begin
                            errmax := 0.0;       {pas tres clair}
                            for j := 1 to n do
                                if errmax < abs(yerr^[j] / yscal[j]) then
                                    errmax := abs(yerr^[j] / yscal[j]);
                        {errmax= 0 therefore this checks if yerrj is not equal to zero due to}
                        { truncation. I still do not see clearly what is done here}
                    errmax := errmax / eps;  {if errmax is effectively zero, it remains so}
        {scale accuracy relative to tolerance}
                            if errmax < 1.0 then
                                begin      {the step has converged}
                                x := x + h;
                                hdid := h;
        {below, we define the big step "hnext" that will be used as "htry" the next}
        {time this routine is called by ODEINT}
                                    if i = nuse then
                                        hnext := h * shrink
                                    else if i = nuse - 1 then
                                        hnext := h * grow
            else {...i is neither nuse nor nuse -1}
            hnext := (h * nseq[nuse - 1]) / nseq[i];
                goto 99 {this is the normal return}
        end        {...of the test checking if the error is OK and defining "hnext"}
                    end            {...of the test (i>3) that permits to enter the OK test}
```

```
        end;            {...of the loop that increases nseqi}
{if we are here, this means that we have reached nseq[imax] = 96 and still}
{do not have enough accuracy to exit. Therefore we must reduce "h" which was}
{previously defined as h:= htry.}
    h := 0.25 * h;
    for i := 1 to (imax - nuse) div 2 do      {i.e. for 1 to 2 do. NOTE "div" is integer "/"}
        h := h / 2;          {idem to h:= h/4. This is a strange "for" because (imax-nuse)/2}
                {may not be an integer depending on imax and nuse definition.}
{overall and because of the values of imax and nuse, we have divided h by 16}
    if x + h = x then
        begin
            writeln('pause in routine BSSTEP');
writeln('step size underflow, h=0'); writeln('press <return> to quit the program'); readln;
            halt;
    {goforit := false;}
            end
        end;  {.. of while goforit = true loop}
99:
    dispose(yerr);
    dispose(yseq);
    dispose(dysav);
    dispose(ysav)
    end;     {... of bsstep}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
    procedure odeint
    (var ystart: RealArrayNVAR;
            n: integer;
x1, x2, eps, h1, hmin: myreal; var nok, nbad: integer;
            var Last_h: myreal;
var OdeintXp: FOROdeintXp; var OdeintYp: FOROdeintYp);
{page 613}
{ystart = a vector of starting values of the functions y's. it is replaced by its value}
{at x2 at the end of the routine OUTPUT}
{n = the size of ystart, or the number of functions y to be integrated simultaneously}
{x1 and x2 = the limit of integration}
{eps = the accuracy: Δo = eps x yscal[i]}
{h1 = a guessed first stepsize}
{hmin = the minimum allowed stepsize}
{nok, nbad = the number of good and bad (but retried and fixed) steps taken. OUTPUT}
{Last_h = the last step used by bsstep before we quit odeint. It will be used as "h1"}
{the next time we call Odeint.}
    label
        99;
    const
        maxstp = 10000;
{the maximum number of steps that will be made to go from x1 to x2}
tiny = 1.0e-30;
        {to be added to a number to compensate for roundoff error}
    var
```

```
        nstp, i: integer;  {food for FOR loop}
            xsav, x, hnext, hdid, h: myreal;
        {xsav = the last x saved. x= is xi}

    {hnext = the estimated next stepsize to carry out next by the integration}
     {subroutine (bsstep) so that the  error is acceptable.  hdid = the step actually }
     {carried out by bsstep.  h= the step suggested to bsstep by odeint. (usually}
     {the previous hnext that bsstep suggested at the end of its work.}
            yscal, y, dydx: ^RealArrayNVAR;
    {yscal = see above: Δo = eps x yscal[i]. The acceptable error Δo will be proportional}
     {to a function of the y given by yscal. This function is defined below.}
            {y and dydx = the vectors of the functions yi, and their derivative dydxi.}
     {odeint will only output y(x2), all the intermediate values y(xi), yscal(xi), and xi}
      {are useless outside of odeint, and are therefore defined as dynamic variables}
    {at the end of the job, ystart will be redefined with the values of y(x2).}
      info: string;    {info is the response of the user to some questions}
        begin
          new(yscal);
          new(y);
          new(dydx);
            {those variable are dynamic, so "new" create their place in memory}
          x := x1;
          if x2 >= x1 then
            h := abs(h1)
          else
            h := -abs(h1);  {make sure we integrate in the right direction}
          nok := 0;
          nbad := 0;
    OdeintKount := 0;  {no step stored}
    for i := 1 to n do
      y^[i] := ystart[i];   {y(x1) is loaded}
    if OdeintKmax > 0 then
      xsav := x - 2.0 * OdeintDxsav;
    {This step is just to insure that the first step will be stored later in the }
    {procedure. The "2.0" is just to insure that x - xsav is larger that OdeintDxsav,}
    {I guess it could be 1.1 or other...}
    for nstp := 1 to maxstp do
    {this is the main loop, which will run from x1 to x2 unless more than maxstp}
      {steps are needed, in which case the program will stop}
      begin
        derivs(x, y^, dydx^);
    {are needed to define yscal, but bsstep will have to recall it for its own use}
    for i := 1 to n do
        yscal^[i] := abs(y^[i]) + abs(dydx^[i] * h) + tiny;
    {this line defines the scaling used to monitor accuracy. When y is large, the}
    {allowed error is proportional to y, but when y comes to zero, the error is }
    {proportional to the Δy represented by the step h. Otherwise, y would never}
     {be able to come to zero because the allowed error would also come to zero. }
     {I think "tiny" is just added to compensate for roundoff error.}
      if OdeintKmax > 0 then     {... we are allowed to store the intermediate result}
    if abs(x - xsav) > abs(OdeintDxsav) then    {...we will store the result}
            if OdeintKount < OdeintKmax - 1 then   {..we still have enough room}
```

A40

```
                    begin
                        OdeintKount := OdeintKount + 1;
                        OdeintXp[OdeintKount] := x;      {we store the x value}
                        for i := 1 to n do
        OdeintYp[i, OdeintKount] := y^[i];   {we store the yi value}
        xsav := x      {xsav represents the last x value that had been stored]}
                    end;               {...of storage process started with "if OdeintKmax > 0"}
        if (x + h - x2) * (x + h - x1) > 0.0 then
    {...the step overshoot the end (x2), so we cut down the stepsize}
            h := x2 - x;
    {%%% here starts the calculations that will make one Δx step. }
    {What we did before was defining the wanted accuracy, save intermediate}
    {results if needed, and check that the step did not overshoot the }
    {end of integration x2.%%%}
bsstep(y^, dydx^, n, x, h, eps, yscal^, hdid, hnext);
        if hdid = h then
    nok := nok + 1 {means we supplied bsstep with an appropriate "h"}
    else
            nbad := nbad + 1;
        if (x - x2) * (x - x1) >= 0.0 then
    {...we are done}
            begin
                for i := 1 to n do
        ystart[i] := y^[i];      {ystart is changed to y(x2)}
    if (OdeintKmax - OdeintKount) > 0 then  {%%%%%de mon cru%%%%}
    {...we save the final step, after checking we still have 1 place left}
                begin
                    OdeintKount := OdeintKount + 1;
                    OdeintXp[OdeintKount] := x;   {we store the last x value}
                    for i := 1 to n do
        OdeintYp[i, OdeintKount] := y^[i]; {we store the last yi value}
        end;                {... of saving the last step}
            Last_h := hnext;        {I want to export the last}
    {step in Odeint so that in its next call corresponding to the next big step}
    {from time_obsi to time_obs(i+1) bsstep will start with an appropriate}
    {first step.}
            goto 99                {normal exit}
            end;                    {...of the test checking if we were done}
            {%%% we continue with the following if we are not done %%%}
    if abs(hnext) < hmin then
                    {...the next h we were supposed to use is too small}
            begin
                writeln('pause in routine ODEINT');
                writeln('stepsize is too small, the program will stop');
                halt;
            end;    {... of test if hnext was getting too small}
        h := hnext;        {This value will be used in the next loop}
        end;                {of the main loop which will turn a maximum of "maxstp" steps}
    writeln('pause in routine ODEINT — too many steps');
    writeln('press <return> to continue and end the routine');
    readln;
```

```
99:
    dispose(dydx);
    dispose(y);
    dispose(yscal)

  end;              {...odeint}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure MetfromTime
(var TimeSample, Met_cal: InVector;
                 data: integer); {INPUT: the initial conditions and constants of the model (global variables),
and the}
{ vector of the times at which observations were made.}
{OUTPUT: the concentrations of methane predicted by the model }
{at the different times at which we took some samples.}
    var
      Last_h: myreal;       {the last step carried out by bsstep in one run of odeint}
      j: integer;
      h1: myreal;           { To enable an update of h1 without modifying}
                    {Odeint_h1}
  begin
    MMin := MetMin(MMinAcc);
    Met[1] := Meto;       {derivs is in my case defined with n=1. Met[1] is used}
{in the procedure to calculate the methane concentrations at all}
{the time steps. Met_cal[1] is only a particular value of Met[1] }
{at the time at which a sample was taken.}
    Met_cal[1] := Meto;   {There is no uncertainty on the first value of Met_cal[1]}
                {NOTE that Met_cal would be a (data, n) matrix if n<>1.}
    tbeg := timeo;       {defines tbeg needed for the first call of odeint}
    h1 := Odeint_h1;
    for j := 1 to data - 1 do
       {call odeint to estimate Met_cal of all observations (a number "data")}
      begin
       nok := 0;
       nbad := 0;      {reset nok and nbad}
       tend := TimeSample[j + 1];
odeint(Met, n, tbeg, tend, Odeint_eps, h1, Odeint_hmin, nok, nbad, Last_h, OdeintXp^, OdeintYp^);
tbeg := tend;
       h1 := Last_h;
         {by doing this I permit Odeint to start directly with an optimum "h" the}
         {next time the routine is called.}
       Met_cal[j + 1] := Met[1];   {Odeint output the value Met[1] which correspond to "tend"}
      end; {.... of the loop to screen all the experimental sampling times}
  end;      {... of MetfromTime}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure TCECellfromMet
(var Met, TCE, Cell: InVector;
                 data: integer); {INPUT the amount of methane, OUTPUT: TCE, Cell corresponding to Met}
{Typically, I use Met_cal, TCE_cal, and Cell_cal.}
    var
```

```
        teta, a, b: myreal;
        i: integer;
    begin
      for i := 1 to data do
        begin
          if Met[i] / Meto <= 0.0 then
            begin
writeln('Met[', i, ']/Meto <= 0.0 in TCECellfromMet the program stops'); halt;
            end;
          teta := (Vl + Hm * Va) * Km * rt / (Vl + Ht * Va) / Kt / rm;
          TCE[i] := TCEo * exp(teta * ln(Met[i] / Meto));
a := (-1 + main / rm / E) * (Met[i] - Meto) + (main * Km / rm / E) * ln(Met[i] / Meto); b := (main / E + rt * tox / Y) *
(Vl + Ht * Va) * Y / Vl / rt * (TCE[i] - TCEo); Cell[i] := Cello + Y * (Hm * Va + Vl) / Vl * a + b;
        end;     {.. of loop over all the data}
    end;       {...TCECellfromMet}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$S CurveFit}
{***************************************************** }
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure DefMetDeriv
(k: integer;
                var lista: IntegerArrayMFIT;
                var a: RealArrayMA;
                mfit: integer;
                var TimeSample, Met_cal: InVector;
                var PDMet: MatDeriv;
                data: integer);
{"lista"= the vector indicating which constant will be varied by MRQMIN. INPUT}
{a= the vector of constants                   INPUT}
{"mfit"= is the quantity of these constants.           INPUT}
{Met_cal, TCE_cal, Cell_cal= the values calculated by the model.   OUTPUT}
{PDMet, PDTCE, PDCell = the matrices of partial derivatives.    OUTPUT}
{Its purpose is to define all the partial derivatives that will be needed for the }
{least square calculation. It would be wasteful to calculate the partial derivatives}
{ separately for each time point, like MRQMIN expects FUNCS to do. Wasteful because}
{to calculate Met_cal[i] numerically, we will have to calculate all the Met_cal[j] for}
{ j<i anyway. So it is better to do the numerical calculation once and store the results.}
{The routine also calculates the values of Met, TCE, and Cell at each time[i].}
{Then FUNCS will simply read the results and give them to MRQMIN.}
{DefAllDeriv will calculate all the derivative NEEDED by MRQMIN. i.e. not the partial}
{derivative with respect to a "constant" constant. Therefore, DefAllDeriv will use }
{the first "mfit" data stored in "lista" to determine which derivatives are needed. Then}
{it will give the value zero to all the partial der. corresponding to "constant" constants.}
{Zero is indeed the correct value because it makes y independent of a variation of one}
{of these constants.}
{The calculations will be made for one set of constants i.e. for one particular vector "a".}
{The routine will COMPLETELY REDEFINE the Km, Kt, rm,... from the values of "a" given}
{as input. Therefore all the following calculations will be made using the values of }
{constants found in "a".}
{main OUTPUT:}
```

```
{Met_cal     the actual values}
{PDMet          the partial derivatives}
    var
        delta: myreal;
        i, j, m, entry: integer;
        VMetPD: ^InVector;
    {VMetPD is needed to calculate the value of one vector of the matrix PDMet.}
    {I can not use PDMet directly in "derivative" because this procedure only }
    {input InVector vectors, and not matrices. Therefore MetPD will be used as}
    {_____%%%%%%%%%%%%%%%%%}
        procedure derivative (var TimeSample, Metcal: InVector;
                    delta: myreal);
    {This procedure assumes that we have already calculated the theoretical values}
    {for methane using the current values for the constants. Those results}
    {are expected to be found in the array Met_cal. Then }
    {the procedure calculates new values for this record, but with one constant}
    {increased of a value (delta*constant). If this constant is XX, then it uses the value }
    {(XX+delta). The record item above can therefore be considered to be 1 vector }
    {RMet(XX). The result of the first steps in this routine is a }
    {new vector whose values are equal to  RMet(XX + delta),}
    {. The second step in the routine consists}
    {in estimating the value of the derivatives of the vector with respect to XX by }
    {dividing their difference by delta, and redefining the vector by their derivatives.}
    {Therefore the OUTPUT of the procedure is the vector of the derivatives.}
        var
            i: integer;
        begin
            if delta = 0.0 then
            begin
    writeln('delta = 0.0 in "derivative"'); halt;
            end;     {I do not want to divide by zero later, better stop now}
            MetfromTime(TimeSample, Metcal, data);
    {MetfromTime makes the enormous job of calculating for each time point a theoretical}
    {value for methane, TCE, and Cell using the constants as they were defined just}
    {before "derivative" was called.}
            for i := 1 to data do
    Metcal[i] := (Metcal[i] - Met_cal[i]) / delta;
            end;     {... of derivative}
    {_____%%%%%%%%%%%%%%%%%}
        procedure TestDelta;
    {its purpose is to test if delta = 0.0.. If it is, probably because one of the }
    {constant is zero, then a default value is assigned to it.}
        begin
            if delta = 0.0 then
            begin
    writeln('constant ', aName[lista[i]], ' = ', a[lista[i]], ' has a delta = zero in DefMetDeriv'); writeln('a default value
    of ', defaultDelta : MW, 'will be assigned to it ');
                delta := defaultDelta;
            end;
        end;     {...of TestDelta}
    {_____%%%%%%%%%%%%%%%%%}
```

```
    begin          {of DefMetDeriv}
      new(VMetPD);
      for i := 1 to numb_obs do
        begin
          for j := 1 to numb_const do
            PDMet[i, j] := 0.0;
          VMetPD^[i] := 0.0;
        end;
      MetfromTime(TimeSample, Met_cal, data);
{Calculate the theoretical values of Met, TCE, and Cell using the }
        {constants defined above.}
      for i := 1 to mfit do
          {will calculate the partial derivative corresponding to the "mfit" first}
          {constants indicated in "lista".}
        begin
{I must input a test because Meto, TCEo, and Cello may have different positions}
{in lista depending on their version. But I know we work with set "k"}
          if lista[i] > 8 then      {... it may represent Hm, Hl, or several Meto, TCEo, Cello, Va,Vl}
            begin
              if (lista[i] = 9) or (lista[i] = 10) then
                begin
                  writeln('ERROR in DefMetDeriv lista points to Hm or Ht');
halt;
        end
else if (lista[i] = SetConst[k, 2]) then entry := 9          {Meto}
else if (lista[i] = SetConst[k, 3]) then entry := 10         {TCEo}
else if (lista[i] = SetConst[k, 4]) then entry := 11         {Cello}
    else if (lista[i] = SetConst[k, 5]) or (lista[i] = SetConst[k, 6]) then
      begin
        writeln('ERROR in DefMetDeriv, lista points to Va or Vl');
        halt;
      end
    else
      begin
writeln('ERROR in DefMetDeriv, lista points to an unknown of this set:'); writeln('lista[', i : 3, '] points to ',
lista[i] : 4, ' and the set # is : ', k : 3); writeln('Vous allez être justement puni YEK, YEK, YEK!!  BOOM!');
        halt;
      end;
  end        {...of if lista >8}
else                  {lista <= 8}
  entry := lista[i];
case entry of
  1:     {Km is a[lista[i]], when lista[i] = 1}
    begin
      delta := a[lista[i]] * RelDelA[lista[i]] * fraction;
      TestDelta;
      Km := a[lista[i]] + delta;
  {I increase Km by a fraction of its previous relative increase}
  { from the last call of MRQMIN. If there has not been an initial call to }
  {MRQMIN, RelDelA[i] will be equal to "FirstAccr". This initialization}
```

```
          {must be done at the beginning of the program.}
          derivative(TimeSample, VMetPD^, delta);
            Km := a[lista[i]];      {give back its original value to Km}
          end;

            2:    {Kt}
              begin
          delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
                Kt := a[lista[i]] + delta;
          derivative(TimeSample, VMetPD^, delta); Kt := a[lista[i]];
              end;
            3:    {rm}
              begin
          delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
                rm := a[lista[i]] + delta;
          derivative(TimeSample, VMetPD^, delta); rm := a[lista[i]];
              end;
            4:    {rt}
              begin
          delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
                rt := a[lista[i]] + delta;
          derivative(TimeSample, VMetPD^, delta); rt := a[lista[i]];   .
              end;
            5:    {main}
              begin
          delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
          main := a[lista[i]] + delta;
          derivative(TimeSample, VMetPD^, delta); main := a[lista[i]];
              end;
            6:    {tox}
              begin
          delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
                tox := a[lista[i]] + delta;
          derivative(TimeSample, VMetPD^, delta); tox := a[lista[i]];
              end;
            7:    {Y}
              begin
          delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
                Y := a[lista[i]] + delta;
          derivative(TimeSample, VMetPD^, delta); Y := a[lista[i]];
              end;
            8:    {E}
              begin
          delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
                E := a[lista[i]] + delta;
          derivative(TimeSample, VMetPD^, delta); E := a[lista[i]];
              end;
            9:    {Meto}
              begin
          delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
          Meto := a[lista[i]] + delta; derivative(TimeSample, VMetPD^, delta);
```

A46

```
  Meto := a[lista[i]];
        end;
      10:    {TCEo}
        begin
  delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
  TCEo := a[lista[i]] + delta; derivative(TimeSample, VMetPD^, delta);
  TCEo := a[lista[i]];
        end;
      11:    {Cello}
        begin
  delta := a[lista[i]] * RelDelA[lista[i]] * fraction; TestDelta;
  Cello := a[lista[i]] + delta; derivative(TimeSample, VMetPD^, delta);
          Cello := a[lista[i]];
        end;
    end;    {... of Case *** of }
    for m := 1 to data do
  { VMetPD will be loaded with partial derivatives corresponding}
  {to constant "i", so I can now define the column "i" of the matrix of partial}
  {derivatives.}
  {NOTE that in this definition, the column # follow "entry", and not the integer}
  {values corresponding to a, aName, aMax, aMin. (at least for integer > 8. }
  PDMet[m, entry] := VMetPD^[m];
        end;    {... of the loop over the "variable" constants of lista.}
  {Concerning the "constant" constant of lista, all the partial diff.}
  {corresponding to them had been set to zero at the beginning of }
  {the procedure.}
      dispose(VMetPD);
    end;    {... DefMetDeriv}
  {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
  {$P}
    procedure PDTCEfromPDMet
  (var PDMet, PDTCE: MatDeriv;
            var Met, TCE: InVector;
            ma, data: integer);
  {INPUT: Met, TCE, PDMet, ma, data}
  {OUTPUT: PDTCE}
  {takes the matrix of partial derivatives PDMet and calculate the corresponding}
  {matrix PDTCE}
      var
        teta: myreal;
        i, m: integer;
    begin
      teta := (Vl + Hm * Va) * Km * rt / (Vl + Ht * Va) / Kt / rm;
  {call TCEfromMet as an alternative if I do not calculate TCE InVector}
  {by writing TCE:= TCEfromMet(Met[i]);}
      for i := 1 to data do
        for m := 1 to ma do
          PDTCE[i, m] := teta * TCE[i] / Met[i] * PDMet[i, m];
    end;      {...PDTCEfromPDMet}
  {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
```

```
{$P}
   procedure PDCellfromPDMet
  (var PDMet, PDCell: MatDeriv;
  var Met, TCE: InVector; ma, data: integer);
```
{INPUT: Met, TCE, PDMet, ma, data}

{OUTPUT: PDCell}

```
{takes the matrix of partial derivatives PDMet and calculate the corresponding}
{matrix PDCell.   Uses equation 21 for dX/dc and dX/da = dX/dc*dc/da}
     var
        dXdC: myreal;
        i, m: integer;
     begin
{I may want to call TCEfromMet as an alternative if I do not calculate TCE InVector}
{by writing TCE:= TCEfromMet(Met[i]);}
     for i := 1 to data do
        begin
dXdC := (Hm * Va + Vl) / Vl * Y * (-1 + main / rm / E + main * Km / rm / E / Met[i] + Km / Kt / rm * (main / E + rt *
tox / Y) * TCE[i] / Met[i]); for m := 1 to ma do
             PDCell[i, m] := dXdC * PDMet[i, m];
        end;      {... of loop over all data}
     end;       {...PDCellfromPDMet}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure OneSetSimpleCalc
(k: integer;
                var a: RealArrayMA;
var TimeSample, Met_cal, TCE_cal, Cell_cal: InVector; var PDMet, PDTCE, PDCell: MatDeriv);
{This procedure will do all the calculations (before a CurveFit) that correspond}
{to ONE set of values "k" with the current values of the vector "a". This is }
{done only using the Simple formula. }
{OUTPUT: Met_cal, TCE_cal, Cell_cal, PDMet, PDTCE, PDCell}
{Remember that Va and Vl, Meto, TCEo,and Cello can vary between sets}
{Also, I do not need to keep in memory PDMet, PDTCE, PDCell, because}
{their results will be compressed in the final PDX matrix that will}
{be exported to mrqmin. For that reason at each "k", those matrices}
     {will be erased by the calculation of the new PDmatrices.}
  begin
     CheckConst(a, aName, ma);
{Will screen all the constants for values <= 0.0 that may crash the program}
  {var a : RealArrayMA }
     {var aName: StringArrayMA;}
     {ma: integer}
     DefKmfromA(k, a);
     {var a: RealArrayMA}
     {Redefine the constants Km, Kt,... from the current values of "a".}
     {"a" will be changed by each step of the least squares.}
     {Even more importantly, this function defines the Meto, TCEo, Cello,}
{Va, Vl needed by each set i.e. as a function of "k"}
     SimpleMetfromTime(TimeSample, Met_cal, SetConst[k, 1]);
     {var TimeSample, Met_cal: InVector;}
```

A48

```
                    {data: integer}
      SimpleTCECellfromMet(Met_cal, TCE_cal, Cell_cal, SetConst[k, 1]); {var Met , TCE , Cell : InVector;    }
                    { data: integer}
            SimpleDefMetDeriv(a, ma, TimeSample, Met_cal, PDMet, SetConst[k, 1]);
                    {var a: RealArrayMA;}
                    {ma: integer;}
                    {var TimeSample, Met_cal: InVector;}
                    {var PDMet: MatDeriv}
      {data: integer}
      {it defines all the derivative and the expected Met, TCE and Cell from}
       {the simple formula.}
      SimplePDCellfromPDMet(PDMet, PDCell, Met_cal, ma, SetConst[k, 1]); {var PDMet , PDCell : MatDeriv;}
      {var Met : InVector;}
       {ma, data: integer}
      PDTCEfromPDMet(PDMet, PDTCE, Met_cal, TCE_cal, ma, SetConst[k, 1]); {var PDMet, PDTCE:
      MatDeriv;}
            {var Met, TCE: InVector;}
            {ma, data: integer}
      {This procedure is the one of the complete model. There is no such}
       {procedure for Simple, because by definition TCE is zero in it.}
      {But of course if I put TCEo =  0.0, I will not get anything from this.}
         end;        {...OneSetSimpleCalc}
      {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
      {$P}
         procedure OneSetCalc
      (k: integer;
               var a: RealArrayMA;
      var TimeSample, Met_cal, TCE_cal, Cell_cal: InVector; var PDMet, PDTCE, PDCell: MatDeriv);
      {This procedure will do all the calculations (before a CurveFit) that correspond}
      {to ONE set of values"k" and with the current values of the constant vector "a".}
      { This is done only using the complete formula. }
      {Remember that Va and Vl, Meto, TCEo,and Cello can vary between sets}
       {Also, I do not need to keep in memory PDMet, PDTCE, PDCell, because}
       {their results will be compressed in the final PDX matrix that will}
       {be exported to mrqmin. For that reason at each "k", those matrices}
       {will be erased by the calculation of the new PDmatrices.}
         begin
           CheckConst(a, aName, ma);
      {Will screen all the constants for values <= 0.0 that may crash the program}
       {var a : RealArrayMA }
            {var aName: StringArrayMA;}
            {ma: integer}
           DefKmfromA(k, a);
            {var a: RealArrayMA}
            {Redefine the constants Km, Kt,... from the current values of "a".}
            {"a" will be changed by each step of the least squares.}
            {Even more importantly, this function defines the Meto, TCEo, Cello,}
      {Va, Vl needed by each set i.e. as a function of "k"}
      {MetfromTime(TimeSample, Met_cal, SetConst[k,1]);}
            {var TimeSample, Met_cal: InVector;}
            {data: integer}
```

```
      {It is useless to call MetfromTime now because DefMetDeriv calls it anyway}
      DefMetDeriv(k, lista, a, mfit, TimeSample, Met_cal, PDMet, SetConst[k, 1]);
             {k: integer}
          {var lista: IntegerArrayMFIT}
          {var a: RealArrayMA;}
          {mfit: integer;}
          {var TimeSample, Met_cal: InVector;}
          {var PDMet: MatDeriv. Remember PDMet will be recreated each time i.e.}
                 {its value will not be kept in memory}
          {data:integer}
   {it defines all the derivative and the expected Met, TCE and Cell from}
    {the simple formula.}
      TCECellfromMet(Met_cal, TCE_cal, Cell_cal, SetConst[k, 1]); {var Met , TCE , Cell : InVector;    }
             { data: integer}
      PDCellfromPDMet(PDMet, PDCell, Met_cal, TCE_cal, ma, SetConst[k, 1]); {var PDMet , PDCell :
      MatDeriv;}
             {var Met, TCE: InVector;}
             {ma, data: integer}
      PDTCEfromPDMet(PDMet, PDTCE, Met_cal, TCE_cal, ma, SetConst[k, 1]); {var PDMet, PDTCE:
      MatDeriv;}
             {var Met, TCE: InVector;}
             {ma, data: integer}
       end;       {...OneSetCalc}
   {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
   {$P}
     procedure DefFuncInput
   (simple: boolean;
                  var a: RealArrayMA;
                  var SetToX: SetInteger3;
                  var SetX: IntegerArrayNDATA;
                  var ndata: integer;  {var because I calculate it}
                  var Xcal: RealArrayNDATA;
                  var PDX: XMatDeriv);
   {Will make all the calculations for all the sets given a certain constant}
   { vector "a"}
   {INPUT: as GLOBAL variable: TimeSample (i.e. for each set); SetConst}
        { (pointers to "a")}
   {simple= if true, I use the simple model, if wrong I use the complete model}
   {a= the vector of all the constants with their current values}
   {SetTox, SetX = are used to transform the calculations results into vectors}
        {of dimensions adequate for mrqmin use. They are defined in}
        {MakeMrqVector.}
   {ndata = the total # of observations input in mrqmin (Met+TCE+CEll)}
   {OUTPUT}
   {Xcal= the vector of the predicted values (it correspond to Xobs, the )}
        {observed values}
   {PDX= the matrix of the PD with respect to the constants}
      var
        c, k, i, t, m, b: integer;
     begin
```

```
        t := 0;
      for k := 1 to numbSet do
        begin
          if simple then
    OneSetSimpleCalc(k, a, TimeSample[k]^, Met_cal^, TCE_cal^, Cell_cal^, PDMet^, PDTCE^, PDCell^) else
              OneSetCalc(k, a, TimeSample[k]^, Met_cal^, TCE_cal^, Cell_cal^, PDMet^, PDTCE^, PDCell^);
 {These procedure calculate everything for ONE set i.e. also Metcal, TCEcal...}
 {Note that they are sensitive to "k" and will automatically choose the right}
 {Meto, TCEo, Cello, Va, and Vl corresponding to "k".}
 {%%%% below, I start creating the INPUT stuff for Mrqmin %%%%%%}
        if k = 1 then
          m := 0
        else
          m := SetToX[k - 1, 3];
 {a test necessary for the first call from i:=1 to SetToX[k,1]}
        for i := m + 1 to SetToX[k, 1] do          {Met data}
          begin
            t := t + 1;
            Xcal[t] := Met_cal^[SetX[t]];
            for c := 1 to 11 do
              PDX[t, c] := PDMet^[SetX[t], c];
          {NOTE that PDMet, PDTCE, and PDCell are precisely defined from }
          {1 to 11: 9, 10, 11 being for Meto, TCEo, Cello. The separations of }
          {PD belonging to different versions of Meto, TCEo, and Cello will}
          {be made in Funcs, not in this routine.}
          end;
        for i := SetToX[k, 1] + 1 to SetToX[k, 2] do    {TCE data}
          begin
            t := t + 1;
            Xcal[t] := TCE_cal^[SetX[t]];
            for c := 1 to 11 do
              PDX[t, c] := PDTCE^[SetX[t], c];
          end;
        for i := SetToX[k, 2] + 1 to SetToX[k, 3] do    {Cell data}
          begin
            t := t + 1;
            Xcal[t] := Cell_cal^[SetX[t]];
            for c := 1 to 11 do
              PDX[t, c] := PDCell^[SetX[t], c];
          end;
      end;        {... of loop K over all sets}
    end;    {... of DefFuncInput}
 {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
 {$P}
  procedure Funcs
 (pos: integer;
              var a: RealArrayMA;
              var Xcal: RealArrayNDATA;
              var PDX: XMatDeriv;
              ma: integer;
```

```
    var yfit: myreal;  {it is a bad mistake not to put "var" : see below}
     var dyda: RealArrayMA);
    {INPUT}
```

{pos = the position of the independent variable (timeX) in its vector. We need}

```
        {to know for each time yfit and dyda are needed, but we only need the }
    {address of this time, not its absolute value. i.e. the absolute value}
     {is time[pos], but we only need "pos" to read in the matrices and InVector}
    {calculated by DefAllDeriv}
    {a= the vector of "constant"}
    {Xcal= the vector of predicted (calculated) values}
    {PDX= the matrix (11 columns ONLY) of partial derivatives}
    {A substantial work of Funcs is to take out of the 9, 10, and 11 columns}
     {of PDX the partial derivatives corresponding to different versions}
        {of Meto, TCEo, Cello.}
    {ma= the number of constants in vector "a"}
    {"ma" is needed because dyda will have the partial derivatives with}
     {respect to ALL versions of Meto, TCEo, and Cello,>>>>>>>>}
    {OUTPUT}


    {yfit= the calculated value at "timeX[pos]". Page 577 there is no "Var" before it, i.e.}
    {this formal parameter would be INTERNAL to the procedure and not modified}
     {by funcs. Therefore yfit would not be output by funcs even though its value}
    {will be needed to calculate yobserved - ycalculated (yfit). This is apparently}
     {an important mistake in the book.}
    {dyda= the vector of partial derivatives of yfit (ONE row of the matrix of P.D.) }
    {with respect to the constants. It is actually a ROW of the matrices of }
     {partial derivatives defined by DefAllDeriv}
    {funcs is the user defined subroutine that will provide mrqmin with the}
    {values predicted by the model, and the partial derivative of the}
     {model with respect to the constants at the "time" time[pos].}
    {Note that DefAllDeriv has only calculated the partial derivatives with respect}
     {to the constants described in "lista", all the other derivatives were set to 0.0.}
        label
          102;
        var
          i, k, myset: integer;
     begin
       for i := 1 to ma do
          dyda[i] := 0.0;
       yfit := Xcal[pos];
       for i := 1 to 8 do
          dyda[i] := PDX[pos, i];
        for k := 1 to numbSet do        {finding in which Set (myset) this data belongs}
         for i := 1 to 3 do
           begin
           {the double loop K, I,  starts with the lowest values of SetToX}
if pos <= SetToX[k, i] then
begin
                 myset := k;
                 goto 102;
```

```
            end;
        end;
    102:
    {SetConst[myset, 2] (for example) gives the adress in the vector "a"}
    {of the version of Meto used by the Set "myset".}
        dyda[SetConst[myset, 2]] := PDX[pos, 9];     {Meto}
        dyda[SetConst[myset, 3]] := PDX[pos, 10];   {TCEo}
        dyda[SetConst[myset, 4]] := PDX[pos, 11];   {Cello}
    end;   {funcs}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
  procedure gaussj
 (var a: RealArrayNPbyNP;
n: integer;
var b: RealArrayNPbyMP; m: integer);
{a= the matrix A (n X n) of A•X=B}
{n= the size of A}
{b= is the matrix B (n X m) of A•X=B.}
{m= the number of columns of B, i.e. 1 in my case}
{This procedure solve the system of linear equations A•X=B. X and B}
{are of same size. }
{The method used is Gauss-Jordan Elimination with full pivoting. The}
{matrix a will be output as its inverse, and b will be output as X, the}
 {results of the equations.}
     var
       big, dum, pivinv: myreal;
i, icol, irow, j, k, l, ll: integer; indxc, indxr, ipiv: ^IntegerArrayNP;
{Those 3 arrays are used for bookkeeping on the pivoting.}
{ipiv= a vector of integer keeping track of which column has already been }
 {processed}
  begin
    new(indxc);
    new(indxr);
    new(ipiv);
    for j := 1 to n do
      ipiv^[j] := 0;
    for i := 1 to n do
      begin          {AAA}
        big :=   0.0;
        for j := 1 to n do
          begin        {BBB        %%not necessary to put begin %%}
            if ipiv^[j] <> 1 then
              for k := 1 to n do
                begin   {CCC, %%not necessary to put begin %%  ipiv^[j] <> 1}
                  if ipiv^[k] = 0 then
                    if abs(a[j, k]) >= big then
                      begin
                        big := abs(a[j, k]);
                        irow := j;
                        icol := k
```

```
                              end
        else if ipiv^[k] > 1 then {abs(a[j,k]) < big}
          begin

    writeln('pause 1 in GAUSSJ - singular matrix'); readln;
                          halt;
                        end
              end     {CCC  for...k, this loop was taken if ipiv^[j] <> 1}
                {Incedently this is the end of the loop searching in one row}
              end;      {BBB   for...j}
                {This is the end of the loop searching in one column}
            ipiv^[icol] := ipiv^[icol] + 1;     {icol = k, see above}
            if irow <> icol then       {...we need to interchange the rows}
                begin        {DDD}
                  for I := 1 to n do
    {This loop changes the row of the matrix A}
                    begin
                      dum := a[irow, I];
    a[irow, I] := a[icol, I]; a[icol, I] := dum
                        end;
                  for I := 1 to m do
    {This loop changes the row of the matrix B}
                    begin
    dum := b[irow, I]; b[irow, I] := b[icol, I]; b[icol, I] := dum
                    end
                end;      {DDD}
            indxr^[i] := irow;
            indxc^[i] := icol;
    {"i" is from loop AAA above. indx r & c keep in memory the position of the }
    {pivot when the loop "i" was made......$$}
    if a[icol, icol] = 0.0 then
              begin
    writeln('pause 2 in GAUSSJ - singular matrix'); readln;
              halt
              end;
    pivinv := 1.0 / a[icol, icol];
            a[icol, icol] := 1.0;
            for I := 1 to n do
              a[icol, I] := a[icol, I] * pivinv;
            for I := 1 to m do
              b[icol, I] := b[icol, I] * pivinv;
            for II := 1 to n do
              if II <> icol then
                begin
    dum := a[II, icol];     { a[II, icol] is the element to set to zero}
    a[II, icol] := 0.0;
                  for I := 1 to n do
    a[II, I] := a[II, I] - a[icol, I] * dum; for I := 1 to m do
      b[II, I] := b[II, I] - b[icol, I] * dum end
            end;         {AAA}
          for I := n downto 1 do
```

A54

```
                if indxr^[l] <> indxc^[l] then
                   for k := 1 to n do
                      begin
                         dum := a[k, indxr^[l]];
                         a[k, indxr^[l]] := a[k, indxc^[l]];
                         a[k, indxc^[l]] := dum
                      end;
              dispose(ipiv);
              dispose(indxr);
              dispose(indxc)
           end;            {...gaussj}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure covsrt
(var covar: RealArrayMAbyMA;
                     ma: integer;
var lista: IntegerArrayMFIT; mfit: integer);
{Page 564. The purpose of this procedure, which is used in MRQMIN, is to transform}
{the mfit X mfit  matrix of covariance into an extended ma X ma matrix}
{of covariance sorted into the proper rows and columns and with zero}
 {variance and covariances set for variables which were held frozen. e.g.}
 {the variance of constant ai will be the element covar[i,i]. The resulting}
 {matrix which is again covar will therefore contain several zeros.}
     var
      j, i: integer;
      swap: myreal;         {temporary storage of element [1,1]}
   begin
     for j := 1 to ma - 1 do
       for i := j + 1 to ma do
          covar[i, j] :=   0.0;
     for i := 1 to mfit - 1 do
        begin
for j := i + 1 to mfit do
            if lista[j] > lista[i] then
   covar[lista[j], lista[i]] := covar[i, j] else
        covar[lista[i], lista[j]] := covar[i, j] end;
     swap := covar[1, 1];
     for j := 1 to ma do
        begin
covar[1, j] := covar[j, j]; covar[j, j] :=   0.0;
        end;
     covar[lista[1], lista[1]] := swap;    {the real position of the old [1,1]}
     for j := 2 to mfit do
covar[lista[j], lista[j]] := covar[1, j];
     for j := 2 to ma do
        for i := 1 to j - 1 do
covar[i, j] := covar[j, i];
     end;            {... covsrt}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
```

```
      procedure mrqmin
     (var x, y, sig: RealArrayNDATA;
     ndata: integer; var a: RealArrayMA; ma: integer;
     var lista: IntegerArrayMFIT; mfit: integer;
     var covar, alpha: RealArrayMAbyMA; var chisq, alamda: myreal);
     {page 577}
     {****INPUT*****        The following variables control the calculations}
     {x= the vector of  independent variable (time).}
     {y= the vector of observations.}
     {sig= the vector of variance (inverse sqrt(weight))}
     {ndata = the number of observations <= ndatap (a constant)}
     {ma= the total number of constants, <= map (a constant)}
     {lista= a vector containing the addresses (integer) of the constants in "a".But only}
     {the first "mfit" ones will be fitted by mrqmin, the remaining "ma" - "mfit"}
      {ones will not be touched by the routine. Lista is a way for the user to }
      {indicate to mrqmin that the experimental data should be used only to fit}
      {certain constants and not all of them.}
     {mfit= the number of elements in vector lista}
     {*****OUTPUT*****          Those variables will be defined inside the procedure}
     {and their purpose is to give the output. i.e. No specific values are needed}
      {at input as these variables will be redefined inside the procedure.}
     {covar = the matrix of covariance that will be used to find the error of the constants.}
      {chisq= the sum of square. If the iteration is succesful, its output value will be}
     {updated to the new value. If the iteration}
     {succeeded, a new value is given to both variables, and if the iteration fails, }
      {"chisq" is reset to the value that MrqminOchisq had before the iteration.}
     {****INPUT/OUTPUT*****          Those variables are changed by the procedure, and}
     {are also controlling the calculation. Therefore unlike the OUTPUT variables,}
      { their input value is also important.}
     {alamda = the lambda factor in Marquardt method. If its value is negative, that}
     {tells Mrqmin it is the first time the procedure is called, and Mrqmin must}
      {therefore read the vector lista. if alamda equal zero, that indicates Mrqmin}
      {that we have reached our solution, and we would like it to compute the}
      {covariance matrix so that we can calculate the error of the estimate in }
      {another routine. Finally, the value of alamda will decrease if the iteration}
      {was succesful, but it will increase if the iteration fail and the new "chisq"}
     . {is larger than the previous one.}
     {a= the vector of the constants. If the iteration is succesful, its output values will be}
     {updated to the new values. But unlike "chisq" its values are needed at input the }
      {next time we call Mrqmin.}
     {alpha= "alpha" as output of Mrqcof is NOT the actual "pseudo-Hessian" which}
     { is obtained by augmenting its diagonal elements by the "trick" of Marquardt.}
      {"alpha" is simply the double product of first derivatives. Moreover, "alpha"}
      {is a mfitXmfit matrix and not maXma.}
     {MrqminBeta= is the vector ((observed-predicted)*∂chi/dconstant.}
     {It is NOT in the interface of the procedure, but is a GLOBAL variable}
      {of the main program.}
     {Each time the procedure mrqmin is called, it makes ONE set of calculation.}
     {It starts with a guess of the constants, which can be either the first}
      {one, or a previous result of a mrqmin run, and compute a new set of }
      {constants. Then the user has to decide if it is needed to make a tighter}
     {estimation of the best fit or to stop. This is the role of the driver "CurveFit".}
      {The vector "lista" which indicates the constants to introduce in the }
```

{calculation need only be read once, the first time mrqmin is called.}
{To indicate to mrqmin that it is the first time, the user will just}
 {give a NEGATIVE VALUE TO ALAMDA, as a SIGNAL. Then the subroutine}
{automatically initialize alamda to 0.001. At the end of the first call}
 {of mrqmin, alamda will be multiplied by 10 if the resulting chisquare}
{is higher than the previous one (i.e. the routine failed to find a minimum),}
 {or divided by 10 if the chisquare is lower. To indicate to mrqmim that no more}
 {calculationS are required and that we just want to get back the covariance}
 {matrix, we set ALAMDA = 0.0.}

```
    label
      99;
    var
      k, kk, j, ihit: integer;
```
{ihit= used in the loops checking the vector lista}
```
    atry, da: ^RealArrayMA;
```
{atry= the new vector of constants obtained by the procedure. It will be the }
 {new values exported only if the new chisq obtained is.lower than the initial one.}
```
    oneda: ^RealArrayMAby1;
```
{oneda= the vector beta given as a one column matrix, to be compatible with }
 {the call of GAUSSJ.}
{____%%%%%%%%%%%%%%}
```
      procedure mrqcof (var x, y, sig: RealArrayNDATA;
                    var a: RealArrayMA;
      var lista: IntegerArrayMFIT; var alpha: RealArrayMAbyMA; var beta: RealArrayMA;
```
{beta = the vector beta is in fact -1/2* gradient (p 573)}
```
                    var chisq: myreal);
```
{All the variables except beta are the same as in the main procedure.}
{The purpose of mrqcof is to evaluate the Hessian matrix (alpha), and }
 {the gradient (to get beta) from the user defined procedure "funcs"}
 {which actually calculates the partial derivatives. }
```
        var
        k, j, i: integer;              {Foodforforloop!}
        ymod, wt, sig2i, dy: myreal;
```
{ymod= the calculated value at "time" x[i]. It will be compared}
 {to its corresponding experimental observation for the evaluation}
 {of the sum of square. Its value is provided by "funcs".}
          {wt= is the weight*one of the partial derivative}
{sig2i = the weight itself}
{dy = the difference between the observed y[i] and }
{theoretical ymod value}
```
        dyda: ^RealArrayMA;
            {vector of partial derivative of ymod with respect to the constants}
      begin
        new(dyda);
        DefFuncInput(simple, a, SetToX, SetX^, ndata, Xcal^, PDX^);
        for j := 1 to mfit do
          begin
```
{initialize alpha and beta to zero. This is necessary for the algorithm below.}
```
            for k := 1 to j do
                alpha[j, k] :=  0.0;
              beta[j] := 0.0
          end;    {... of "j"}
```

```
                chisq := 0.0;
                for i := 1 to ndata do
                  begin
                        Funcs(i, a, Xcal^, PDX^, ma, ymod, dyda^);
                        sig2i := 1.0 / (sig[i] * sig[i]);   {define the weight of data # i}
                        dy := y[i] - ymod;          {le "vi"}
                        for j := 1 to mfit do
                          begin
                            wt := dyda^[lista[j]] * sig2i;  {dyda w.r.to one of the constant}
                            for k := 1 to j do    {the columns 1 to j}
                              alpha[j, k] := alpha[j, k] + wt * dyda^[lista[k]];
                            beta[j] := beta[j] + dy * wt;
                          end;         {... of "j" the summation which partially defined }
                              {half of alpha and all beta}
                        chisq := chisq + dy * dy * sig2i     {partially define the sum of square}
                  end;                      {... of "i" the summation over all the experimental data}
                for j := 2 to mfit do              {the columns 2 to mfit}
                      {define the other half of matrix alpha, the right of the diagonal}
                  for k := 1 to j - 1 do             {the rows 1 to j-1}
                    alpha[k, j] := alpha[j, k];
                dispose(dyda)
              end;     {... of internal routine mrqcof}
      {_____%%%%%%%%%%%%%%%}
        begin       {... of mrqmin}
        new(da);
        new(oneda);  {a matrix with one column, the vector "da", used by GaussJ}
        new(atry);
        if alamda < 0.0 then
            begin     { AAA if this is the first time we run the routine, we must initialize}
                {the vector of constants to compute}
            kk := mfit + 1;
            for j := 1 to ma do
                {screen lista over all its length. Its role is mainly to check if }
{the vector lista has been properly defined by the user. In particular}
{that the address of the same constant is not given twice}
                begin     {BBB}
                    ihit := 0;
for k := 1 to mfit do
{screen the the elements [1..mfit] of lista to determine if "j" belongs}
{to them.}
if lista[k] = j then    {... "j" is a "variable" constant}
ihit := ihit + 1;
if ihit = 0 then
{..."j" is a "constant" constant by default. Its address therefore belongs}
{to the last [mfit..ma] elements of lista.}
begin
                    lista[kk] := j;
                    kk := kk + 1
                  end               {... ihit = 0}
else if ihit > 1 then
```

```
{There are two elements of lista with the address "j". This is not acceptable}
  begin
writeln('pause 1 in routine MRQMIN'); writeln('Improper permutation in LISTA');
                readln;
                halt
              end
          end;  {BBB}
        if kk <> ma + 1 then
{kk > ma+1 if there are less "variable" constants in lista than "ma"}
  {in that case the loop BBB automatically give more "constant" }
  {constants than it should}
          begin
            writeln('pause 2 in routine MRQMIN');
            writeln('Improper permutation in LISTA');
readln;
halt
          end;
        alamda := 0.001;
        mrqcof(x, y, sig, a, lista, alpha, MrqminBeta, chisq);
        Mychisq := chisq;      {de mon cru. Used internally to CurveFit}
        MrqminOchisq := chisq;
      end; {AAA}
{#######The steps above are only done the first call of mrqmin.}
{The next steps are the actual calculations.#####}
    for j := 1 to ma do
  atry^[j] := a[j]; for j := 1 to mfit do
begin   {CCC}
      for k := 1 to mfit do
  covar[j, k] := alpha[j, k]; covar[j, j] := alpha[j, j] * (1.0 + alamda);
{covar is the Hessian matrix as modified by the Marquardt method}
oneda^[j, 1] := MrqminBeta[j]
      end;  {CCC}
    gaussj(covar, mfit, oneda^, 1);
{gaussj will solve the system: covar•X=oneda for X on output}
  {covar is replaced by the matrix inverse and oneda by X}
    for j := 1 to mfit do
      da^[j] := oneda^[j, 1];      {the accroissement vector}
    if alamda = 0.0 then
      begin
covsrt(covar, ma, lista, mfit);
      goto 99
      end;     {...of IF alamda = 0}
    for j := 1 to mfit do
      begin
        atry^[lista[j]] := a[lista[j]] + da^[j];
{I test if the constant proposed in atry are within the limits of }
{aMin and aMax.}
if ((atry^[lista[j]] <= 0.0) or (atry^[lista[j]] <= aMin[lista[j]]) or (atry^[lista[j]] >= aMax[lista[j]])) and not
((atry^[lista[j]] = 0.0) and (aMin[lista[j]] < 0.0)) then begin
            writeln('atry of ', aName[lista[j]] : 5, ' is ', atry^[lista[j]] : MW);
            writeln('a ', a[lista[j]] : MW);
```

```
if a[lista[j]] = 0.0 then
   RelDelA[lista[j]] := (atry^[lista[j]] - a[lista[j]]) * FirstAccr else
              RelDelA[lista[j]] := (atry^[lista[j]] - a[lista[j]]) / a[lista[j]];
       end;   {...of defining the constants to try}
         mrqcof(x, y, sig, atry^, lista, covar, da^, chisq);
         if chisq < MrqminOchisq then
           begin
              alamda := 0.1 * alamda;
MrqminOchisq := chisq; for j := 1 to mfit do
begin
                for k := 1 to mfit do
   alpha[j, k] := covar[j, k]; MrqminBeta[j] := da^[j]; a[lista[j]] := atry^[lista[j]]
                end
           end
         else
           begin
              alamda := 10.0 * alamda;
              chisq := MrqminOchisq
           end;
99:
dispose(atry);
dispose(oneda);
dispose(da)
       end;    {... of mrqmin}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}
   procedure CurveFit
(var a: RealArrayMA);
{The purpose of CurveFit is to drive Mrqmin. This is necessary because Mrqmin}
{only makes one iteration. Its role will be first to provide Mrqmin with the }
{necessary inputs which are the same throughout the iterations, then }
{throughout the iterations to re-give the input/output to the procedure (this}
{is done simply by recalling the procedure with the same interface). In addition,}
{by analyzing the convergence of "chisq" towards a minimum, decide when we}
{have reached it with enough accuracy.}
   label
     101;
   var
     word: string[12];
     iteration: integer;
begin
   alamda := -1.0;
     {to indicate to Mrqmin that this is the first iteration}
   for iteration := 1 to itermax do
     begin
        mrqmin(TimeX^, Xobs^, Xsig^, ndata, a, ma, lista, mfit, covar^, alpha^, chisq, alamda);
        if Mychisq > chisq then        {.. the step was successful}
          begin
             Mychisq := chisq;
             writeln('in CurveFit, iteration ', iteration : 3, ' alamda = ', alamda : MW, 'chisq = ', chisq : MW);
```

```
                    if not screen then        {I want this info together with OUTconstant in the file}
      writeln(sort, 'in CurveFit, iteration ', iteration : 3, 'alamda = ', alamda : MW, 'chisq = ', chisq : MW);
      OUTconstants(screen, sort, a, aName, ma, MW, DP);
              {Will print the values of the constant vector a on the screen if "true"}
              {or in a file previously opened if "false".}
      writeln('press any key to continue, "n" to stop the iteration');
                  readln(word);
                  if (word = 'n') or (word = 'N') then
                      goto 101;
              end;     { of if the step was successful}
{it the step is not successful, then nothing is displayed and the program tries again}
          end;     {... of iterations }
{if we reach this point it is because we did "itermax" iterations}
writeln('more than ', iteration : 3, ' iterations in CurveFit. STOP');
      halt;
101:
{normal exit}
      alamda := 0.0;
mrqmin(TimeX^, Xobs^, Xsig^, ndata, a, ma, lista, mfit, covar^, alpha^, chisq, alamda); {Here Mrqmin is only
used to compute the covar matrix}
      end;       {...CurveFit}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
 {%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{$P}


begin       {beginning of the main program.}
  oldFreeHeap := FreeMem;
  oldStack := StackSpace;     {Needed by procedure HeapStack}
{NOTE the pointers to TimeSample[k] are defined in a loop in "Define"}
  HeapStack('Beginning of program');
  new(Met_obs);
  new(TCE_obs);
  new(Cell_obs);
  new(MetSig);
  new(TCESig);
  new(CellSig);
  new(Met_cal);
  new(TCE_cal);
  new(Cell_cal);
  new(PDMet);
  new(PDTCE);
  new(PDCell);
  new(covar);
  new(alpha);
  new(OdeintXp);
  new(OdeintYp);
  new(SetX);
  new(TimeX);
  new(Xobs);
  new(Xcal);
```

```
       new(Xsig);
       new(PDX);

     HeapStack('end of main program');
       Initialize;
    {Call all the procedure necessary to read the input file, and initialize}
     {the constants needed by the program}
       CurveFit(a);
         {interactive procedure, will find the best estimated of a}
     writeln('after AllCalculations, Methane PD');
       OUTAllSetObsCal(TimeX^, Xobs^, Xcal^, Xsig^, TimeSample, SetX^, SetConst, SetToX);
    {var TimeX, Xobs, Xcal, Xsig: RealArrayNDATA;}
     {var TimeSample :SetInVector}
         {SetX: IntegerArrayNDATA;}
         {SetConst: SetInteger6;}
         {SetToX: SetInteger3}
       OUTconstants(screen, sort, a, aName, ma, MW, DP);
    {Will print the values of the constant vector a on the screen if "true"}
     {or in a file that will be named like "title" if "false".}
         {screen: boolean;}
         {var sort: text; }
    {var a: RealArrayMA;}
    {var aName: StringArrayMA;}
    {var ma, MW, DP: integer);}
     for k := 1 to numbset do
       begin
      dispose(TimeSample[k]); end;          {of dispose loop K}
      dispose(Met_obs);
      dispose(TCE_obs);
      dispose(Cell_obs);
      dispose(MetSig);
      dispose(TCESig);
      dispose(CellSig);
      dispose(Met_cal);
      dispose(TCE_cal);
      dispose(Cell_cal);
      dispose(PDMet);
      dispose(PDTCE);
      dispose(PDCell);
      dispose(covar);
      dispose(alpha);
      dispose(OdeintXp);
      dispose(OdeintYp);
      dispose(SetX);
      dispose(TimeX);
      dispose(Xobs);
      dispose(Xcal);
      dispose(Xsig);
      dispose(PDX);
    end.
```

A62

Appendix B
Notation

C Methane or growth substrate concentration in the aqueous phase, mg/L.

C_o Initial methane concentration, g/L.

E Mass of enzymatically active sMMO per mass unit of microbial cell, or (for Model 3) fraction of cells that are metabolically active for degrading either the growth substrate (C) or the cometabolic substrate (T), unitless.

ECD Electron capture detector.

E_{tot} Total enzyme concentration, g/mL.

F Additional constant or groups of constants needed by some of the models (given without description of their meaning).

FID Flame ionization detector.

G Additional constant or groups of constants needed by some of the models (given without description of their meaning).

GAC Granular activated carbon.

GC Gas chromatograph.

H Additional constant or groups of constants needed by some of the models (given without description of their meaning).

H_C Henry's constant of methane, unitless.

H_T Henry's constant of TCE, unitless.

J Additional constant or groups of constants needed by some of the models (given without description of their meaning).

K_C	Half-saturation constant for methane, mg/L.
K_O	Half-saturation constant for oxygen, mg/L.
K_T	Half-saturation constant for TCE, mg/L.
$M1$	A pure culture of methane-oxidizing bacteria.
$M2$	A pure culture of methane-oxidizing bacteria.
$M3$	A pure culture of methane-oxidizing bacteria.
$M4$	A pure culture of methane-oxidizing bacteria.
$M5$	A pure culture of methane-oxidizing bacteria.
Ma	A mixed culture of methane-oxidizing bacteria.
Mb	A mixed culture of methane-oxidizing bacteria.
Mc	A mixed culture of methane-oxidizing bacteria.
MCB	Methylococcus capsulatus (Bath), a pure culture of methane-oxidizing bacteria.
MMO	Methane mono-oxygenase.
Mx	A pure culture of methane-oxidizing bacteria.
O	Oxygen concentration in the aqueous phase, mg/L.
PHB	Poly-hydroxybutyrate.
$pMMO$	Particulate methane mono-oxygenase.
r_C	Rate constant for methane, L/hr.
r_O	Rate constant for oxygen, L/hr.
r_T	Rate constant for TCE, L/hr.
$sMMO$	Soluble methane mono-oxygenase.
T	Concentration of TCE or substrate of cometabolism in the aqueous phase, mg/L.
t	Time, hr.
TCA	1,1,1-trichloroethane.

TCE	Trichloroethylene.
T_o	Initial TCE concentration, mg/L.
Total SWS	Total sum of weighted squares.
V_a	Volume of air in contact with the liquid, mL.
V_l	Volume of liquid, mL.
X	Concentration of microbial cells in the aqueous phase, mg/L.
X_o	Initial cell concentration, g/L.
Y	Cell yield: mass of cells created by unit mass of methane metabolized for growth, unitless.
β	Maintenance or decay constant, unitless.
ξ	Toxicity constant: mass of cells killed per unit mass of cometabolic substrate oxidized, or additional demand exerted by cometabolism on cell metabolism, unitless.

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE February 1995	3. REPORT TYPE AND DATES COVERED Final report

4. TITLE AND SUBTITLE
Microbial Degradation of Volatile Anthropogenic Organic Chemicals

5. FUNDING NUMBERS

6. AUTHOR(S)
Martin Alexander, Francois Roch

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Soil, Crop, and Atmospheric Sciences
Cornell University
Ithaca, NY 14853

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Engineer Waterways Experiment Station
3909 Halls Ferry Road, Vicksburg, MS 39180-6199

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Miscellaneous Paper EL-95-2

11. SUPPLEMENTARY NOTES
Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Experiments were conducted to study the degradation of trichloroethylene (TCE) by bacteria able to grow on methane (methanotrophs) and to consider specific aspects relative to the ultimate design of a bioreactor to purify air streams contaminated with TCE that could originate from air stripping of contaminated aquifers.

A procedure was investigated that consisted of initially sorbing TCE from the gas phase to granular activated carbon (GAC). The GAC then was treated by first extracting TCE from the GAC by using methanol and then providing the methanol containing TCE to methanotrophs. The experiments indicated that neither TCE nor methane could be significantly degraded by methanotrophs in the presence of a high but nontoxic concentration of methanol in water.

A study was conducted to determine whether there is a concentration of TCE (a threshold) below which methanotrophs growing on methane would not be able to degrade TCE. TCE was degraded below a concentration of about 2 parts per trillion, and thus no threshold was found.

The degradation of TCE by methanotrophs in the presence of different packing materials was assessed. The results showed that some packing materials inhibited TCE degradation unless they were first washed with an aqueous solution of ethylenediaminetetraacetic acid (EDTA).

(Continued)

14. SUBJECT TERMS
Degradation
Granular activated carbon
Methane
Methanotrophs
Trichloroethylene

15. NUMBER OF PAGES
124

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

13. (Concluded).

A mathematical model was derived to simulate the biodegradation of TCE by methanotrophs growing on methane. One purpose of the model is to help in the design of a bioreactor to purify air streams contaminated with low concentrations of TCE. A computer program was written for that purpose.